Universidad Politécnica de Cartagena
Departamento de Electrónica, Tecnología de Computadoras y Proyectos

Diseño e implementación sobre hardware reconfigurable de una arquitectura para la emulación en tiempo real de redes neuronales celulares

José Javier Martínez Álvarez

2012

Universidad Politécnica de Cartagena
Departamento de Electrónica, Tecnología de Computadoras y Proyectos

Diseño e implementación sobre hardware reconfigurable de una arquitectura para la emulación en tiempo real de redes neuronales celulares

José Javier Martínez Álvarez

Director

José Manuel Ferrández Vicente

2012

A mis padres

Agradecimientos

Quiero agradecer a José Manuel Ferrández su apoyo y paciencia durante el desarrollo de esta tesis, sobre todo en los momentos bajos en los que nunca ha puesto pegas ni negativas, todo lo contrario. Mi agradecimiento sincero por el ánimo y el trato de amistad recibido.

También quiere agradecer a todos los miembros del Departamento de Electrónica Tecnología de Computadores y Proyectos de la Universidad Politécnica de Cartagena el apoyo recibido durante estos años y especialmente a mis compañeros y amigos los Javis con los cuales he compartido incontables horas de trabajo y multitud de experiencias inolvidables.

Resumen

En esta Tesis se propone el diseño y la implementación sobre hardware reconfigurable de una arquitectura para la emulación en tiempo real de redes neuronales celulares (CNN). El proceso de diseño de la arquitectura, comienza con el planteamiento de diferentes métodos de discretización del modelo continuo original de la red CNN. A partir de dichos métodos se obtienen distintas aproximaciones que son simuladas y comparadas entre sí con el fin de comprobar su funcionalidad y determinar cuál de ellas proporciona los mejores resultados con el menor coste computacional. La aproximación con mejores prestaciones es elegida para desarrollar el algoritmo de cómputo que describe la arquitectura hardware de la red CNN.

La metodología de desarrollo utilizada, explora diferentes alternativas para optimizar la arquitectura CNN desde el punto de vista de su implementación hardware sobre FPGAs. A partir de la paralelización y adaptación del algoritmo de cómputo se desarrollan dos arquitecturas hardware diferentes denominadas Carthago y Carthagonova. Estas arquitecturas describen el funcionamiento de una Celda CNN, desenrollada en Etapas, que permite emular secuencialmente el procesamiento realizado por las redes CNN. La principal característica de estas arquitecturas es la capacidad que tienen para procesar la información en flujo de datos y en tiempo real. Las soluciones propuestas tiene como principal objetivo conseguir el mejor equilibrio entre la velocidad de procesamiento y el consumo de recursos hardware de la FPGA, así como evitar el uso de dispositivos de memoria externa que reducen la velocidad de procesamiento del sistema e incrementan su tamaño.

Se proponen diferentes alternativas para implementar las arquitecturas sobre dispositivos FPGAs. Una de ellas consiste en utilizar una técnica de sincronización *self-timed*, eficiente en área-tiempo, que es definida mediante un lenguaje de descripción hardware tradicional (VHDL), instanciando primitivas de bajo ni-

vel y realizando el emplazamiento de los componentes de forma manual. Otra alternativa consiste en una descripción en VHDL estructural a nivel RTL y sincronización convencional, donde los componentes *self-timed* son sustituidos por componentes estándar. Se propone además la implementación de una de las arquitecturas sobre un computador reconfigurable de altas prestaciones (HPRC), compuesto por un microprocesador de propósito general y un coprocesador basado en FPGAs, encargado de acelerar la ejecución de los algoritmos mediante hardware. El particionamiento hardware/software y el proceso de co-diseño se realizan usando las herramientas de desarrollo a nivel de sistema (ESL) de Impulse Accelerated Technologies (Impulse-C) y la plataforma HPRC DS1002 de DRC Computers.

Los principales resultados obtenidos de las diferentes implementaciones son mostrados con el fin de demostrar la funcionalidad de las arquitecturas y analizar sus principales prestaciones. Las diferentes combinaciones consideradas, entre técnicas de implementación y las arquitecturas propuestas, muestran que la arquitectura Carthagonova, implementada a nivel estructural, presenta importantes ventajas a considerar. En primer lugar, la arquitectura facilita la emulación de redes CNN complejas, compuestas por cientos de miles de millones de neuronas, sobre sistemas embebidos basados en FPGAs. En segundo lugar, el excelente compromiso alcanzado entre velocidad de procesamiento y consumo de recursos hardware hace que sea una interesante solución a considerar frente a otras alternativas de la literatura. Finalmente, la versatilidad y las prestaciones de la arquitectura diseñada permiten dar soporte al desarrollo de sistemas de procesamiento de vídeo en tiempo real y al diseño de aplicaciones basadas en modelos neuronales bioinspirados.

La arquitectura CNN propuesta es utilizada para desarrollar un modelo artificial de la primera sinapsis de la retina, incorporando algunas de las principales características de los circuitos neuronales considerados. El modelo está basado en los campos receptores de las células bipolares y su objetivo es emular, mediante hardware reconfigurable, el procesamiento espacial básico realizado por la retina. Al igual que ocurre en la primera sinapsis de la retina, se observa que el modelo artificial propuesto lleva a cabo la detección del contraste y la discriminación visual de detalles en función de la influencia de los factores de convergencia y de inhibición lateral de los circuitos neuronales implementados.

Finalmente, se propone el diseño y la implementación de un sistema de cómpu-

to distribuido, basado en múltiples FPGAs, que permite el desarrollo de aplicaciones embebidas de procesamiento de vídeo en tiempo real con redes CNN multicapa (ML-CNNs) complejas y de gran tamaño. El sistema procesa la información de vídeo en flujo de datos (en modo progresivo) y proporciona una salida de vídeo estándar compatible con el formato VGA industrial.

Las principales ventajas del sistema vienen determinadas por las características de la arquitectura de cómputo distribuida utilizada. La arquitectura está basada en múltiples Módulos de Procesamiento (MP) interconectados entre sí, que permiten la expansión del sistema y su adaptación a nuevas aplicaciones. Los MP son conectados en cascada, mediante una interfaz de expansión compartida, a una fuente de vídeo de entrada y a un *frame grabber*, encargado de adaptar la información de vídeo procesada al formato de salida del sistema. Cada MP está compuesto por una placa FPGA que aloja a la red CNN o parte de esta. La capacidad de cómputo del sistema vendrá determinada por la cantidad de MP utilizados y por la cantidad de recursos hardware disponibles en las FPGAs. Los componentes pre-diseñados de la arquitectura CNN, junto con la modularidad y ampliabilidad del sistema, proporcionan un excelente entorno de desarrollo de aplicaciones de procesamiento de vídeo en tiempo real basadas en redes CNN complejas y de gran tamaño.

Abstract

This thesis proposes the design and development of a hardware architecture for real-time emulation of non-linear multilayer Cellular Neural Networks (CNN). This approach is focused on CNN implementation on reconfigurable hardware architectures. The architecture design begins from a discrete model obtained from different transformations made to the original continuous model of CNN. Each discrete approach is simulated and compared with the rest of approaches in order to verify their functionality and to find the approach which best emulates the continuous model at minimum computational cost. The best discrete model found is then used to develop the hardware architecture of the CNN.

The development methodology used, explores different alternatives to optimize the architecture from the point of view of its hardware implementation on FPGAs. The architectures Carthago and Carthagonova are developed from the hardware adaptation and parallelization of the sequential algorithm that describes the functionality of the selected CNN discrete model. These architectures are based on an unrolling cell which is employed to emulate CNNs with large number of neurons. The key characteristic of these architectures is their capability to process information in real time in a sequential manner. The proposed solution aims to find a suitable tradeoff between area and speed, reducing the use of hardware resources on the FPGA and avoiding the use of external memory devices which make slower the processing rate and higher size and cost.

We propose different solutions to the internal implementation of both architectures on an FPGA. The first one is a novel self-timed architecture, area-time efficient. It is described using traditional Hardware Description Languages (HDL) from low level hardware primitives instantiation and manual placement. The second one consists in a high level description in structural VHDL using conventional synchronization, instead of self-timed blocks. We also propose an implementation architecture that makes use of a High Performance Reconfigura-

ble Computer (HPRC), combining general purpose microprocessors with custom hardware accelerators based on FPGAs, to speed up execution time. The hardware/software partitioning and co-design process are carried out using high level design tools. The architecture Carthago is implemented using Electronic System Level (ESL) tools from Impulse Accelerated Technologies and the DS1002 HPRC platform from DRC Computers.

The most relevant results obtained from different implementations are shown in order to verify the functionality of the proposed neural hardware architectures and to analyze their performance. The best combination of architecture and implementation model, the Carthagonova-structural, presents some important advantages. Firstly, the architecture is developed to help emulating high-performance discrete CNNs with hundreds or millions of neurons on embedded FPGA-based systems. Secondly, due to its balanced trade-off between speed and area, this architecture is an interesting alternative to consider among others in the literature. Finally, its versatility facilitates the export of neural hardware architecture to applications such as signal and image processing, implementation of the neurological models inspired on human systems, etc.

The hardware architecture proposed is used to build a CNN-based model of the fist synapse of the retina, which incorporates the main neural circuits found in the different retinal regions. The aim of this bioinspired model is to implement the basic spatial processing of the retina in reconfigurable hardware. The model is based on the bipolar cells receptive fields and mimics the retinal architecture achieving its processing capabilities. As occurs in the processing of first synapse of the retina, it is observed that contrast detection and detail resolution are influenced by the convergence factor of neurons and by lateral inhibition, which are specific parameters of each neural circuit.

We also propose the design and implementation of an embedded system based on multiple FPGAs that can be used to process real time video streams for applications that require the use of large Multi-Layer CNNs (ML-CNNs). The system processes video in progressive mode and provides a standard VGA output format.

The main features of the system are determined by using a distributed computing architecture, based on Processing Modules (PM), which facilitates system expansion and adaptation to new applications. Several FPGA-based processing modules can be cascaded together with a video acquisition stage and an output

interface to a frame grabber for video output storage, all sharing a common communication interface. Each PM is composed by an FPGA board that can hold one or more CNN layers. The total computing capacity of the system is determined by the number of MP used and the amount of resources available in the FPGAs. The pre-verified CNN components, the modular architecture, and the expandable hardware platform provide an excellent workbench for fast and confident developing of CNN applications, based on traditional cloned templates, but also time-variant and space-variant templates.

Índice general

Índice de cuadros

Índice de figuras

Preámbulo

La bioingeniería es un campo de estudio en constante desarrollo. Esta disciplina conjuga los conocimientos fundamentales de la medicina y la biología con materias tecnológicas específicas como la electrónica, la informática, la óptica y la robótica para aplicar técnicas e ideas de ingeniería a determinados problemas de la medicina y de la biología.

Una de las principales lineas de investigación en bioingeniería es el desarrollo de sistemas bioinspirados. Con estos sistemas se trata de comprender cómo los sistemas biológicos trabajan, aprenden y evolucionan para posteriormente encontrar la manera de imitarlos. La aspiración de los investigadores es conseguir circuitos electrónicos dotados con sentidos artificiales similares a los del ser humano (oído, olfato, vista, etc.), o capaces de imitar su funcionamiento, con el fin de resolver problemas hasta ahora no resueltos y con ello facilitar la vida a personas discapacitadas.

Para tratar de emular las funciones sensitivas de los seres humanos, como la visión, la capacidad olfativa o el reconocimiento del habla, es necesario contar al menos con los siguientes elementos: un modelo biológico y artificial capaz de describir el funcionamiento del sistema a emular, componentes tecnológicos con la suficiente capacidad de cómputo como para permitir reproducir el procesamiento del sistema artificial, y un equipo interdisciplinar, formado por físicos, informáticos, electrónicos, médicos, biólogos, fisiológos, etc. donde la formación, experiencia y colaboración de sus miembros contribuya a la consecución de los objetivos.

Desarrollo de sistemas bioinspirados

En campos como la física y la ingeniería, las ecuaciones diferenciales y algebraicas son utilizadas para describir el comportamiento de sistemas reales y desarrollar modelos matemáticos que los representen. El interés por modelar un sistema biológico radica en la posibilidad de utilizar dicho modelo para reproducir su funcionamiento y predecir determinados fenómenos. El diseño de un modelo artificial, capaz de ser implementado electrónicamente y de imitar el comportamiento sensitivo del ser humano, requiere de un modelo biológico de partida cuyo desarrollo, sin embargo, plantea múltiples e importantes retos difíciles de resolver.

Los sistemas biológicos reales, a partir de los cuales se obtienen los modelos biológicos, son entornos complejos, dinámicos e impredecibles que presentan multitud de incógnitas difíciles de resolver. Si analizamos los componentes involucrados en la obtención de un modelo biológico, observaremos que su desarrollo requiere de un conocimiento profundo del sistema real y de un conjunto de técnicas, herramientas y métodos de cálculos de alta complejidad. No obstante, es importante tener en cuenta que el objetivo de muchos modelos no es conseguir un comportamiento perfectamente idéntico al del sistema, sino más bien reproducir únicamente aquellas propiedades de interés que puedan resultar de utilidad para las aplicaciones.

Para llevar a cabo el desarrollo de un modelo biológico es necesario tener en cuenta varios aspectos. En primer lugar, se ha de estudiar todos y cada uno de los conceptos relevantes que forman parte del sistema biológico real. En segundo lugar, las variables que caracterizan dichos conceptos, y que tendrán que ser medidas, deberán ser verdaderos indicadores del fenómeno a analizar. En tercer lugar, el proceso de medición de las variables deberá cuantificar realmente los conceptos a medir. Finalmente, para validar los modelos obtenidos, será necesario realizar un análisis comparativo y crítico de los resultados obtenidos por el sistema real y por su modelo. Todo este proceso deberá realizarse con el instrumental adecuado, para garantizar la calidad y validez de la información adquirida.

El principal problema que presenta el desarrollo de un modelo biológico radica en la gran cantidad de datos que se han de medir y que, en la mayoría de los casos desafortunadamente, se encuentran dispersos sobre una gran variedad de señales fisiológicas. Para intentar desvelar cómo estas señales codifican la información es

necesario utilizar instrumental de precisión que, sin embargo, en general es demasiado simple comparado con la complejidad de los sistemas biológicos. Dicho instrumental proporcionará información incompleta, dado que únicamente podrá medir un número limitado de variables, que además en muchos casos será imprecisa e incluso ambigua, debido a la gran complejidad del proceso de medición, lo que dificultará la obtención de modelos realistas de los sistemas.

Los modelos artificiales utilizados por los sistemas bioinspirados están basados en sus homólogos biológicos, los cuales, previamente, son simplificados mediante técnicas matemáticas de aproximación, linealización, discretización, etc. con el fin de hacer factible su implementación. Dichas transformaciones, sin embargo, introducen mayores errores e incertidumbres en los modelos que ocasionan respuestas aún más alejadas de las deseadas.

El desarrollo de un sistema bioinspirado conlleva la búsqueda de un modelo artificial con un compromiso adecuado entre precisión y sencillez que garantice, en el entorno en el que se desea que funcione, la operatividad del sistema y su implementabilidad.

Modelado del sistemas nervioso

A pesar de las dificultades que entraña el desarrollo de sistema biosinspirados, la emulación de sistemas biológicos ha sido desde siempre uno de los principales retos de la ciencia. El sistema biológico que tradicionalmente más interés ha despertado, por su complejidad y capacidad para proporcionar recursos adaptativos a los sistemas bioinspirados, ha sido el sistema nervioso humano.

El sistema nervioso está formado por millones de estructuras neuronales que desempeñan multitud de funciones con características y aptitudes inmejorables (Jain, 1996): robustez, tolerancia a fallos, capacidad de aprendizaje, adaptación al medio, manejo de información difusa, procesamiento paralelo, bajo consumo, tamaño reducido, etc. Todas ellas cualidades deseables para cualquier tipo de sistema. Sin embargo, conseguir un modelo de la complejidad del sistema nervioso, formado por cientos de millones de neuronas altamente no lineales e interrelacionadas entre sí, es en la actualidad prácticamente imposible. De lo que se trata, por tanto, no es de emular a la perfección su comportamiento, sino más bien de

imitar como realiza el procesamiento y adoptar esa habilidad para diseñar sistemas con la capacidad de predecir determinados comportamientos a partir de un aprendizaje.

Se quiere dejar claro, por tanto, que las redes neuronales biológicas son demasiado complejas como para ser modeladas por simples expresiones matemáticas, lo que subyace bajo estas, es que para abordar los problemas que el cerebro resuelve de manera eficiente puede ser interesante tratar de imitar su estructura y comportamiento.

Redes neuronales artificiales

Las redes neuronales artificiales (RNA) son un paradigma creado por el hombre para emular, de forma simplificada, la estructura y el comportamiento funcional del sistema nervioso. Las RNA combinan unidades de procesamiento simples, que almacenan datos de la experiencia y del aprendizaje en forma de refuerzo entre sus conexiones (pesos). Aunque es posible encontrar gran variedad de RNA, con características y propiedades diferentes, todas ellas tienen en común el hecho de que su estructura y funcionamiento tienen como base de inspiración la estructura neuronal del sistema nervioso.

En el escenario más simple, una neurona biológica es una célula nerviosa especializada en transmitir y modular la información que trascurre por el sistema nervioso. Dichas células están constituidas por un soma, que aloja el núcleo de la célula, varias ramificaciones denominadas dendritas, que actúan como vías de entrada de los estímulos de la célula, y una prolongación alargada y ramificada llamada axón, que propaga los impulsos nerviosos y que establece las conexiones con otras neuronas, a través de sus dendritas.

Aunque en el sistema nervioso existen multitud de neuronas diferentes, con distintas formas, ramificaciones, longitudes de axón, etc., todas procesan la información de forma similar. Dentro de la neurona, las señales se propagan como impulsos eléctricos, denominados potenciales de acción, que se generan tras la combinación de numerosos estímulos en las entradas de la célula. Cuando el potencial de acción llega a los terminales del axón, la neurona libera un conjunto de neurotransmisores químicos, que a su vez, estimulan las dendritas de otras

neuronas, estableciéndose lo que se denomina sinapsis neuronal.

Esta descripción simple del funcionamiento de una neurona biológica representa la principal fuente de inspiración de las redes neuronales artificiales. A continuación se resumen algunas de las principales características compartidas tanto por las RNA como por las redes neuronales biológicas:

- Cada neurona recibe múltiples señales de entrada.
- La señal recibida puede ser modificada en función de los pesos de la sinapsis.
- La neurona realiza la suma ponderada de las entradas.
- Bajo condiciones adecuadas, la neurona trasmite una sola salida.
- La salida de una neurona se puede distribuir a muchas otras neuronas.
- La información se procesa localmente.
- La memoria está distribuida (la memoria consiste en una matriz de pesos).
- El peso de cada sinapsis se modifica tras un aprendizaje.
- Los pesos pueden ser excitadores o inhibidores.

En definitiva, las RNA pueden ser vistas como modelos matemáticos simples que, bajo ciertas condiciones operativas, pueden ser utilizadas para resolver determinados problemas de procesamiento. En general, su capacidad de aprendizaje y habilidad para establecer relaciones entre conjuntos de entradas y salidas son su principal atractivo.

El estudio de las RNA ha sido abordado por multitud de investigadores abarcando diferentes campos de interés y multitud de aplicaciones, principalmente relacionadas con problemas de clasificación, reconocimiento de patrones, control, procesamiento de imágenes, etc. A pesar de que las RNA son técnicas muy utilizadas, su alto coste computacional exige, en ocasiones, que sea necesario acelerar su ejecución mediante hardware específico.

Las RNA comenzaron a ser desarrolladas en los años 70 con la principal motivación de entender cómo funciona el sistema nervioso y tratar de emularlo. Desde

el principio, las RNA surgieron como una prometedora herramienta capaz de aportar soluciones interesantes a multitud de aplicaciones y áreas de la ciencia. Entre sus principales virtudes destacan la posibilidad de aprendizaje, a partir de nuevas experiencias, la mejora de su desempeño y la adaptación a cambios en el entorno.

El primer modelo de una neurona artificial, denominado perceptrón, fue desarrollado en la década de los 60 por Rosenblatt (Rosenblatt, 1962). Dicho modelo despertó gran interés debido a su capacidad para aprender a reconocer patrones sencillos, aunque sólo permitía la discriminación de patrones linealmente separables. Otros modelos de la época, las redes adaline (*adaptive linear element*) y madeline (múltiple adaline) podían ser entrenadas mediante métodos de mínimos cuadrados. La red adaline fue la primera red neuronal artificial que fue aplicada a problemas reales (Anderson, 1992), utilizándose como filtro adaptativo para la eliminación ecos en líneas telefónicas.

En la siguiente década, el interés por las RNA decayó tras la publicación de un libro de Minsky y Papert (Minsky, 1969), donde se demostraba matemáticamente que el perceptrón no podía resolver determinadas funciones lógicas. Holpfield (Hopfield, 1982), junto con la aparición del algoritmo de aprendizaje *backpropagation*, propuesto inicialmente por Werbos (Werbos, 1974) y posteriormente reinventado por Rumelhart et al (Rumelhart, 1986), devolvieron la popularidad a las RNA.

Conforme la neurobiología ha ido evolucionando más y más modelos de RNA han salido a la luz, siendo algunos de los más representativos: los mapas autoorganizativos de Kohonen (SOM, *selft-organization maps*) (Kohonen, 1972; Kohonen, 1982), la teoría de la resonancia adaptativa (ART, *Adaptive resonance theroy*) (Grossberg, 1976), las redes de función de base radial (RBF, *Radial Basic Funtion*) (Broomhead, 1988) y sobre todo, el perceptrón multicapa (MLP, *Muli-Layer Perceptron*).

Las RNA más simples son sistemas estáticos cuya respuesta no depende del tiempo, es decir, la salida es calculada a partir de las entradas actuales sin tener en cuenta los valores de pasado. En estas condiciones, las neuronas artificiales no disponen de variables de estado interna que influyan en su respuesta. Se ha podido comprobar, sin embargo, que lo habitual en el sistema nerviosos es que los

circuitos neuronales estén realimentados, es decir, las salidas de las neuronas están conectadas a sus propias entradas de forma directa, o bien de forma indirecta a través de otras neuronas. La existencia de realimentación en las redes neuronales convierte a estas en sistemas dinámicos complejos y considerablemente más difíciles de analizar.

Cuando una neurona realimentada recibe una entrada, la salida correspondiente es enviada nuevamente a la entrada que a su vez genera una nueva salida. Este proceso cíclico se repetirá hasta que la salida alcance un valor constante, es decir un estado estable, o se repetirá de forma indefinida si el sistema es oscilante y su estado inestable. La presencia de realimentación en una red neuronal puede afectar en gran medida a su capacidad de aprendizaje. Algunos de los tipos de RNA realimentadas más conocidas son las redes de Hopfield, la máquina de Boltzmann y las redes neuronales celulares (CNN, *Cellular Neural Network*)

Las redes CNN, objeto de esta tesis doctoral, fueron propuestas por Chua y Yang en 1988 (Chua L., 1988) y representan un caso particular de las RNA de Hopfield. Las redes CNN son utilizadas principalmente en aplicaciones de procesamiento de imagen y mimetizan algunas de las características básicas de las redes neuronales del sistema nervioso, como la realimentación y el procesamiento local de la información. Estas redes fueron creadas como solución robusta y de bajo consumo a la implementación de las RNA sobre circuitos electrónicos. El interés por este tipo de redes ha sido tal que en 1990 se propuso la organización de un congreso internacional (CNNA), patrocinada por IEEE, dedicado en exclusiva a este tipo de redes y a sus aplicaciones, celebrandose en el año 2012 su décima tercera edición.

En general, el reciente interés por las redes CNN puede ser atribuido a dos factores diferentes. El primero relacionado con el desarrollo de nuevas técnicas de aprendizaje, que permiten aprovechar mejor las características de estas redes y llevar a cabo problemas no resueltos hasta ahora. El segundo relacionado con los avances en computación, donde concretamente la computación paralela permite aprovechar eficientemente las características de las redes CNN y ahondar de manera más ágil en su estudio y desarrollo.

Implementación hardware de las redes neuronales artificiales

Se estima que en el cerebro humano existen aproximadamente cien mil millones de neuronas interconectadas entre sí que funcionan en armonía para resolver problemas específicos. De este increíble entramado de células nerviosas es interesante destacar, como una de sus principales característica, su capacidad para trabajar en paralelo. A partir de esta consideración es razonable pensar que las RNA, inspiradas en el sistema nervioso, también pueden obtener beneficio de la computación paralela.

Existen multitud de desarrollos que tratan de explotar el paralelismo de las RNA a partir del uso de computadores convencionales basados en múltiples procesadores. Fijándonos en las características de estos computadores (basados en la arquitectura Von Neumman), podemos observar que sus prestaciones han crecido considerablemente en los últimos años. Dicho crecimiento permite que muchas aplicaciones basadas en RNA encuentren en la simulación software la solución más acertada a la implementación, principalmente cuando los requisitos computacionales no son exigentes o cuando no es requerido un procesamiento en tiempo real. La principal ventaja de la simulación software es la versatilidad que ofrece a la hora de la experimentación, dada su capacidad para modificar los sistemas neuronales de forma rápida y sencilla. El cuadro 1, extraído de (Jain, 1996), muestra las principales diferencias entre las arquitecturas Von Neumann y los sistemas neuronales biológicos.

La utilización de hardware específico como solución estándar al desarrollo de RNA no es todavía una solución madura debido principalmente a las dificultades que supone su realización. Sin embargo, cuando las aplicaciones RNA requieren capacidades de computo superiores a las ofrecidas por las plataformas software, el diseño hardware específico es la única solución factible. En aplicaciones concretas, como la emulación de sistemas bioinspirados, otra razón que favorece la utilización de hardware específico es la posibilidad de diseñar sistemas embebidos, cuyas menores dimensiones facilitan la adaptación de los sistemas bioinspirados al entorno en el que se desea que funcionen.

En la implementación hardware de un sistema neuronal se pueden distinguir

dos fases. Una independiente de la tecnología empleada, que consiste en especificar las características y requisitos del modelo neuronal y de las aplicaciones. Y otra relacionada con la tecnología, encargada de seleccionar y diseñar los aspectos relacionados con la plataforma de implementación.

En el caso de que el sistema neuronal se implemente mediante hardware específico, se tendrá un mayor grado de libertad para el diseño pero también se requerirá mayor tiempo de desarrollo y un conocimiento más profundo del problema a resolver. Por otra parte, si la solución se ha planteado para hacer uso de hardware de terceros, las especificaciones del sistema neuronal deberán ser más generales y posiblemente surgirán mayores dificultades a la hora de adaptar el problema al hardware utilizado.

Cuadro 1: Arquitectura Von Neumann contra Sistemas neuronales biológicos

	Arquitectura Von Neumann	Sistema neuronal biológico
Procesador	Complejo	Simple
	Alta velocidad	Baja velocidad
	Una unidad o pocas	Gran número
Memoria	Separada del procesador	Integrada en el procesador
	Localizada	Distribuida
	Contenido no direccionable	Contenido direccionable
Operaciones	Centralizadas	Distribuidos
	Secuenciales	Paralelos
fiabilidad	Muy vulnerable	Robusto
habilidad	Numérica y simbólica	Problemas de percepción

Implementación on-Chip de redes neuronales artificiales

La implementación de sistemas neuronales on-Chip es vista como una solución muy atractiva que permite alcanzar niveles de funcionamiento y prestaciones muy superiores a las conseguidas por otras plataformas. Sin embargo, la utilización de Sistemas-on-Chip (SoC) como solución alternativa a la implementación de RNA aún no ha alcanzado un nivel de madurez satisfactorio debido a las grandes dificultades que supone su realización. Según Dean R. Colins, director del laboratorio

de componentes de sistemas de Texas Instruments (Collins, 1989), el "*ideal neural network chip*" debería contar con las siguientes características.

- Trabajar con cualquier algoritmo neuronal.
- Tener aproximadamente 1000 neuronas por chip.
- Interconexión rápida y programable.
- Interconexiones analógicas.
- Interconexionado global entre neuronas.
- Que las operaciones de las neuronas se realicen en microsegundos.
- Bajo coste por neurona.
- Bajo consumo de potencia (necesario si el sistema tiene un número considerable de neuronas).
- Baja ocupación de área por neurona (para que el chip sea pequeño).
- Interfaz I/O acondicionadas (pocos pines con suficiente ancho de banda).
- Estable.
- Reproducible.
- Trabajar a temperatura ambiente.
- Escalable (en el sentido de poder construir mosaicos con otros chip similares en altura y anchura o quizás incluso en tres dimensiones).

Requisitos que, aunque fueron enunciados hace más dos décadas, aún en día continúan siendo difíciles de conseguir.

El diseño de una red neuronal on-Chip suele comenzar con la elección del modelo neuronal que se desea implementar, el cual generalmente dependerá del tipo de problema a resolver. Para concretar el modelo y su estructura (número de neurona, capas, interconexión, etc.) será necesario también conocer la información que se desea procesar y si esta requiere preprocesamiento o no. Cuando la

RNA demanda el uso de tecnología analógica, la solución más apropiada, por sus inmejorables prestaciones en cuanto a velocidad de funcionamiento y consumo de área, es la basada en circuitos integrados de aplicación específica (ASIC, *Application Specific Integrated Circuits*). Si las técnicas de implementación son digitales, además de utilizar ASIC digitales, se podrán emplear dispositivos hardware reconfigurables como las FPGA (*Field Programmable Gate Array*). El cuadro 2 resume las principales características de las diferentes soluciones on-Chip usadas para implementar RNA.

Cuadro 2: Características principales de las diferentes tecnologías usadas en la implementación de sistemas neuronales artificiales.

Tecnología Empleada	Dificultad de Implementar	Velocidad	Systema en tiempo real	Consumo de potencia
ASIC Analog.	Alta	Muy alta	Si	Bajo
ASIC Digital	Baja	Alta	Posible	Medio bajo
FPGA	Medio baja	Media	Posible	Medio alto
DSP (software)	Baja	Medio alto	Posible	Alto

La realización completa de una RNA supone que todas las entidades de procesamiento (neuronas) son implementadas físicamente, es decir, que cada una de ellas hace uso exclusivo de su propio circuito. La mayoría de realizaciones de este tipo únicamente puede ser implementadas sobre circuitos ASIC analógicos, debido al menor consumo de área de los circuitos. Por el contrario en las implementaciones digitales, donde las exigencias de espacio son mucho mayores, lo habitual es que las neuronas compartan las mismas unidades funcionales que serán multiplexadas en el tiempo. En este caso, el proceso de diseño deberá considerar el equilibrio adecuado entre la velocidad de funcionamiento y el consumo de área con el fin de conseguir una implementación eficiente.

Los principales inconvenientes que plantea la realización de las RNA sobre circuitos integrados radican en la complejidad que supone el diseño de los propios circuitos y el tiempo de desarrollo. En todas las técnicas de fabricación de circuitos integrados VLSI (*Very Large Scale Integration*), excepto en el caso de las FPGAs, la modificación de los circuitos implica la fabricación completa del dispositivo, lo que supone varios meses de trabajo. En este sentido, la principal

ventaja de las FPGAs, con respecto a las otras técnicas de implementación, reside en la posibilidad de llevar a cabo la reconfiguración de los circuitos de manera inmediata.

Por el motivo anterior, quizás, la plataforma más prometedora para implementar las RNA digitales sea la FPGA. Estos dispositivos cuentan con una estructura interna regular, similar a la de una memoria, que permite aprovechar los últimos avances en fabricación para integrar grandes cantidades de bloques lógicos configurables. Las funciones lógicas de estos bloques y su interconexión, definidas por el usuario, son cargadas en el chip de manera similar a como se almacenan los datos en una memoria. Las FPGAs actuales incorporan la funcionalidad de millones de puertas lógicas capaces de funcionar a frecuencia que superan los 500 Mhz. Por otro lado, existen multitud de herramientas y entornos de diseño que, de manera sencilla y automática, permiten transformar la descripción de alto nivel de circuitos electrónicos en ficheros de configuración de la FPGA.

Motivación y Objetivos

El grupo de investigación de "Diseño electrónico y técnicas de tratamiento de señales" de la Universidad Politécnica de Cartagena junto con la unidad de "Fundamentos de la Visión y Visión Artificial" y el departamento de Histología y Anatomía del Instituto de Bioingeniería de la universidad de Elche están colaborando en una línea de investigación común que pretende el desarrollo de un prototipo de retina artificial neuromórfica que sirva de ayuda a personas con discapacidad visual o con pérdida total de la visión.

En el marco de este trabajo se propone la realización de una tesis doctoral cuyo objetivo es realizar el diseño, desarrollo y la implementación electrónica sobre hardware reconfigurable de una arquitectura capaz de emular en tiempo real el comportamiento de las redes Celular Neural Network (CNN). Estas redes tienen gran interés en el campo de la bioingeniería debido a sus adecuadas características para desarrollar aplicaciones de procesamiento de imagen y sistemas bioinspirados tales como las retinas artificiales.

Para comprobar la funcionalidad y eficiencia de nuestra propuesta, la arquitectura hardware de la red CNN será implementada y aplicada al desarrollo de

un modelo artificial de la primera sinapsis de la retina. Se propone además la realización de un sistema de cómputo distribuido que posibilite el desarrollo, rápido y sencillo, de aplicaciones en tiempo real basadas en redes CNN complejas y de gran tamaño.

A continuación se desglosan los objetivos específicos que se desarrollarán con el fin de alcanzar el objetivo genérico previsto en el presente trabajo:

1. Analizar los requisitos de diseño, desarrollo e implementación de una arquitectura hardware de propósito general que permita dar solución al mayor número posible de aplicaciones basadas en redes CNN, en particular aquellas relacionadas con el procesamiento de imágenes de vídeo en tiempo real. Para llevar a cabo esta tarea se pondrá especial interés en tratar de superar las limitaciones encontradas en otras alternativas y de enfocar el diseño hacia una solución que sea eficiente en área y velocidad.

2. Explorar diferentes modelos y aproximaciones que permitan optimizar la arquitectura desde el punto de vista de su implementación hardware, así como plantear una metodología con la que proyectar las expresiones de las distintas alternativas propuestas sobre hardware reconfigurable. La concreción de dichas arquitecturas serán traducidas a hardware usando diferentes técnicas y niveles de descripción (de bajo y alto nivel), que abrirán el abanico de posibilidades de implementación sobre diversas plataformas.

3. Desarrollar una arquitectura hardware basada en redes CNN que permita el desarrollo de aplicaciones de procesamiento de imágenes y vídeo en tiempo real, de forma jerárquica y estructurada, preservando las características esenciales de sencillez, modularidad, versatilidad y escalabilidad. La arquitectura será evaluada desde el punto de vista de su funcionalidad, realizando simulaciones que permitan comprobar su operatividad, y también desde el punto de vista de la eficiencia de su implementación hardware, considerando como factores clave a evaluar: el consumo de recursos hardware y la velocidad de funcionamiento.

4. Desarrollar, a partir de la arquitectura propuesta, un modelo de la primera sinapsis de la retina que incorpore sus principales circuitos neuronales y que permita emular su procesamiento espacial básico. Dicha aplicación servirá

para verificar la metodología de diseño y la versatilidad y flexibilidad de la arquitectura hardware que se propone.

5. Implementar un sistema de cómputo distribuido, compuesto por múltiples FPGAs, que permita evaluar y verificar la funcionalidad de la arquitectura desarrollada bajo condiciones reales de funcionamiento. Las principales características del sistema estarán determinadas por su versatilidad y modularidad, las cuales deberán facilitar la ampliación del sistema y su adaptación a nuevas aplicaciones. Se pretende que el sistema pueda ser utilizado como banco de pruebas y plataforma de desarrollo rápido de aplicaciones basadas en redes CNN complejas y de gran tamaño.

Estructura de la Memoria

El trabajo se ha estructurado en 5 capítulos, además del presente preámbulo y de las conclusiones.

El Capítulo 1 realiza una introducción al paradigma de la redes neuronales celulares, analizando los fundamentos y conceptos básicos del modelo y de su arquitectura clásica. Se describe las características de la celda estándar y las variantes del modelo original más utilizadas.

El Capítulo 2 aborda las diferentes soluciones utilizadas hasta la fecha para implementar redes CNN. Se analizan las características de las principales tecnologías de fabricación de circuitos integrados y se profundiza en las realizaciones hardware de las redes CNN tanto analógicas como digitales, haciendo hincapié en las arquitecturas más importantes: describiendo su funcionamiento, analizando su evolución temporal e indicando sus principales características.

El Capítulo 3 introduce las especificaciones y requisitos que serán considerados en el diseño de la arquitectura de la red CNN. En primer lugar se definen el marco de trabajo que simplifica la arquitectura y que permite especificar la funcionalidad del sistema desarrollado para verificar el funcionamiento de la arquitectura. A continuación, se define lo que entenderemos por arquitectura CNN, los niveles de abstracción y paralelismo utilizados en el diseño y los requisitos generales y reglas de diseño considerados en su desarrollo. Se revisan las princi-

pales arquitecturas de cómputo paralelas y se describe la metodología de diseño utilizada para desarrollar la arquitectura CNN. Finalmente, se proponen diferentes alternativas de discretización del modelo de la red CNN con el objetivo de encontrar la aproximación más adecuada para la implementación hardware.

El Capítulo 4 describe en detalle las soluciones hardware que se proponen: dos arquitecturas modulares que permiten el desarrollo de aplicaciones basadas en redes CNN de forma jerárquica y organizada. Se proponen también diferentes alternativas para la implementación de la arquitectura interna, usando diferentes lenguajes de descripción hardware y plataformas de implementación. Los resultados obtenidos por las diferentes implementaciones son evaluados y comparados, exponiéndose los resultados principales y características más ventajosas de las distintas soluciones.

Por último, en el Capítulo 5 la arquitectura y la metodología de diseño propuestas son validadas mediante el desarrollo de un modelo artificial de la primera sinapsis de la retina y de un sistema de cómputo distribuido, modular y versátil, diseñado para ser usado como plataforma de desarrollo y prototipado rápido de aplicaciones basadas en redes CNN.

Capítulo 1

Redes Neuronales Celulares

Las Redes Neuronales Celulares (CNN) representan un paradigma análogo a la computación paralela. La capacidad para procesar y propagar la información las hace especialmente útiles en problemas donde la información necesaria para su resolución aparece contenida en torno a un punto en consideración. Dichos problemas tienen la particularidad de que todos los puntos de su dominio pueden ser desarrollados de forma simultánea.

Las redes CNN fueron concebidas como estructuras neuronales recurrentes de carácter no lineal cuya complejidad dinámica, definida a partir de un conjunto de ecuaciones diferenciales sencillas, proporciona un comportamiento global complejo que las cataloga como redes neuronales dinámicas. Su estructura multidimensional, constituida por un conjunto de celdas idénticas localmente interconectadas, favorece el desarrollo y la realización de sistemas neuronales On-Chip de grandes dimensiones.

Desde la aparición de las redes CNN, la base matemática subyacente en su modelo ha sido ampliamente estudiada, proporcionando importantes resultados de investigación que han favorecido su desarrollo y aplicación. Las redes CNN han sido utilizadas de manera exitosa, tanto en aplicaciones de tiempo continuo como de tiempo discreto, para resolver problemas de reconocimiento de patrones, seguimiento de objetivos, inspección visual, visión inteligente, sistemas bioinspirados, etc. Su adaptación al hardware y el tipo de aplicaciones a las que se destinan

han sido las principales razones que han motivado el desarrollo de la arquitectura hardware que se propone en esta tesis doctoral.

En este capítulo se abordarán los conceptos básicos y características principales de las redes CNN. Se proporcionará una visión general de su arquitectura estándar y de las variantes más utilizadas. El objetivo que se prende es sentar las bases para el desarrollo de los siguientes capítulos.

1.1. Introducción a las redes CNN

Las redes CNN fueron concebidas por Chua y Yang en 1988 (Chua L., 1988) como un nuevo paradigma de la computación neuronal. La principal característica de este tipo de RNA radica en la utilización de un modelo computacional recurrente en el que solamente es posible el intercambió local de la información. Dicho modelo considera que únicamente es posible interconectar neuronas cercanas entre sí, lo que, sin embargo, no impide que neuronas lejanas también puedan intercambiar información. Para ello, la propia estructura de la red es utilizada como canal de transmisión y propagación de la información. Este mecanismo permite que efectos producidos en determinados puntos de la red puedan alcanzar cualquier otra parte de la red sin necesidad de establecer una conexión directa. Durante el proceso de propagación, la información generada por las neuronas es combinada en el interior de la red, produciéndose complejas interacciones dinámicas de carácter global. Dichas interacciones, generalmente vinculadas a una función de salida no lineal, pueden llegar a depender incluso de todas las entradas y salidas de la red.

El origen de este modelo, dinámicamente complejo, surgió tras la búsqueda de una alternativa a las redes clásicas de Hopfield (Hopfield, 1982) que proporcionase una solución factible al problema de la implementación hardware de las RNA. Con dicho objetivo, las redes CNN fueron concebidas como estructuras uniformes, basada en circuitos idénticos, donde únicamente es posible la interconexión local de las neuronas cercanas.

Las redes CNN pueden ser definidas formalmente de la siguiente manera (Manganaro, 1999): "*Una red neuronal celular (CNN) es un circuito no lineal de alto orden dinámico compuesto por unidades recurrentes, denominadas celdas, que se*

interconectan localmente en el espacio, constituyendo diferentes topologías: rectangular, hexagonal, toroidal, esférica, etc. y que pueden ser definidas matemáticamente a partir de cuatro especificaciones:"

- *La dinámica de la celda.*
- *La ley sináptica.*
- *Las condiciones de contorno.*
- *Las condiciones iniciales*

1.1.1. Arquitectura estándar de la red CNN

Las redes CNN pueden ser representadas mediante sistemas de celdas definidos en el espacio normalizado R^n. La dimensión de estos sistemas no suele exceder $n > 3$ debido a la dificultad que supone su fabricación sobre dispositivos integrados analógicos, generalmente basados en procesos planares. En el espacio bidimensional, las celdas de la redes CNN pueden ser mapeadas sobre diferentes esquemas físicos, como, por ejemplo, los mostrados en la figura 1.1. Con independencia de la topología empleada, las redes CNN pueden estar formadas por celdas idénticas entre sí o por un conjunto reducido de celdas diferentes, como ocurre en el caso de la redes NUP-CNN (*Non-Uniform-Processor* CNN), figuras 1.1-d y 1.1-f.

Como se ha indicado anteriormente, las celdas de las redes CNN únicamente pueden estar conectadas a celdas que estén situadas en su proximidad, es decir, en su entorno de vecindad. La vecindad de una celda se define como el conjunto de celdas que se sitúan a su alrededor y sobre las cuales esta ejerce su influencia de forma directa. Según esta definición es posible encontrar una función distancia que determine exactamente la región de vecindad de cada celda. En general, esta región suele venir representada mediante una esfera, centrada en la propia celda, y cuyo radio r especifica el tamaño de la vecindad. Hay que mencionar que existen redes CNN con diferentes tamaños de vecindad que se denominan MNS-CNN (*Multiple-Neighborhood-Size* CNN) (Roska T., 1992). Un ejemplo de este tipo de redes, constituida por celdas con radio de vecindad $r = 1$ y por celdas con radio de vecindad $r = 2$, se muestra en la figura 1.1-e.

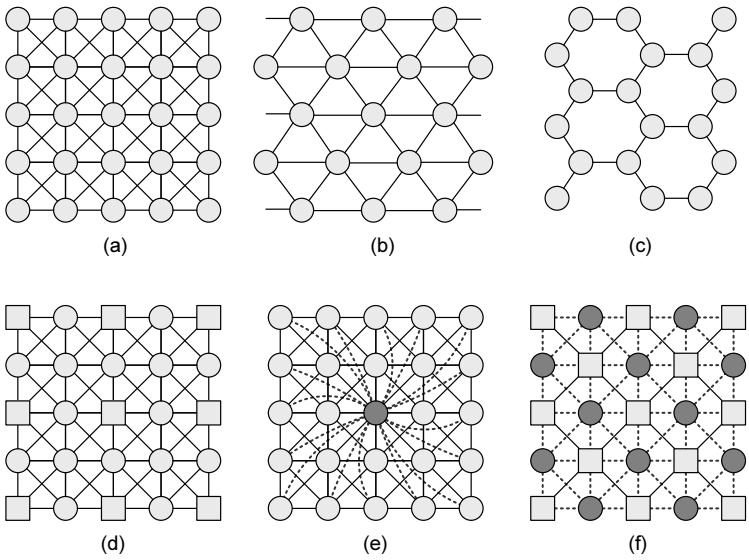


Figura 1.1: Ejemplos de redes CNN con distintas topologías y conexiado. a) Rectangular. b) Triangular. c) Hexagonal. d) Diferentes celdas (NUP-CNN). e) Diferente tamaño de vecindad (MNS-CNN). f) Diferentes celdas y conexionado.

Como se ha visto, las redes CNN pueden presentar multitud de topologías y conexionados diferentes. En la práctica, sin embargo, lo habitual es que las redes CNN formen estructuras bidimensionales de una sola capa y radios de vecindad reducidos. La arquitectura más estandarizada de las redes CNN es representada mediante una matriz de celdas, de dimensión M×N, que se distribuyen uniformemente sobre el plano. Dicha distribución hace uso de un sistema de coordenadas cartesiano, de variables discretas (i, j) (siendo $i = 1, 2, ...M$ y $j = 1, 2, ..., N$), que permite ordenar las celdas en filas y columnas, según un patrón geométrico rectangular. Como se observa en la figura 1.2, a partir de este esquema, las celdas pueden ser identificadas fácilmente mediante sus correspondientes coordenadas cartesianas $C(i, j)$. La forma rectangular de esta distribución, similar al formato de una imagen, permite asociar fácilmente las celdas de la red a los píxeles de la imagen, lo que propicia que sea la topología más utilizada en procesamiento de imágenes.

En la arquitectura estándar de las redes CNN, la vecindad de las celdas se define mediante la expresión $(2r+1)\times(2r+1)$, cuya representación gráfica coincide con un cuadrado de lado $2r + 1$ centrado en la celda $C(i, j)$, siendo r el radio de vecindad de la celda. En la figura 1.1-a se muestra la arquitectura estándar de

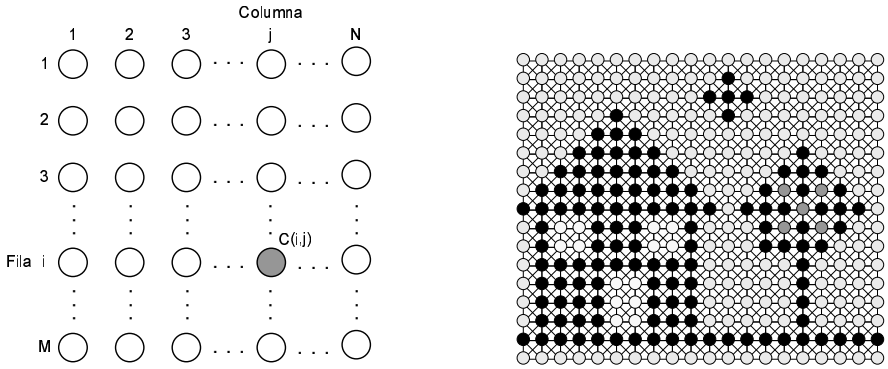


Figura 1.2: Distribución estándar de una red CNN de M×N celdas.

la red CNN para un radio de vecindad $r = 1$. En dicha figura se observa como las celdas se conectan a sus 8 vecinas de alrededor para dar lugar a una ventana de vecindad de tamaño 3×3. Para un radio de vecindad $r = 2$ (celda central de la figura 1.1-e), las celdas se conectan con sus 24 vecinas obteniéndose una ventana de tamaño 5×5. De forma general, para un radio de vecindad r dado, el conjunto de celdas que forman vecindad con la celda $C(i, j)$ vendrá definido por la expresión siguiente.

$$N_r(i,j) = \{C(k,l)| \ \ max\left(|k-i|, |l-j|\right) \leq r\}, \quad \forall \quad \begin{smallmatrix} 1 \leq k \leq M \\ 1 \leq l \leq N \end{smallmatrix} \tag{1.1}$$

Una caso particular es cuando el radio de vecindad de la red CNN es mayor o igual al tamaño de la propia red ($r \geq M/2$, suponiendo redes cuadradas $M = N$). En ese caso todas las celdas de la red estarán interconectadas entre sí, lo que da lugar a redes CNN completamente conectadas. En esta situación, la interconexión de las redes CNN coincide con el esquema de las redes clásicas de Hopfield y, por tanto, la implementación hardware resulta más costosa y difícil de realizar. En estas circunstancias el número de líneas de conexionado crece exponencialmente con el tamaño de la red y la dificultad de realizar redes de gran tamaño aumenta considerablemente. A pesar de este inconveniente, hay que indicar que existen implementaciones físicas de estas redes, entre las que destaca el chip comercial ETANN (Intel, 1991) fabricado por Intel, que integra una red de Hopfield de 64 neuronas con un total de 10.240 conexiones sinápticas.

Es interesante destacar que las interacciones que se producen en el interior de las redes CNN, entre sus celdas, no dependen ni del tamaño de la red ni de la posición de las celdas en su interior. Dichas interacciones dependen principalmente

de una matriz de pesos, denominada matriz de coeficientes o plantilla (*template* en ingles), que determina la intensidad de las conexiones sinápticas entre las celdas y la propagación de la información a lo largo de la red. De forma equivalente a como sucede en las conexiones sinápticas de las redes neuronales biológicas, los elementos de esta matriz establecen la contribución de cada entrada en la respuesta de la celda y determina la fuerza con la que se conectan las celdas entre sí. Para caracterizar los pesos de la redes CNN suelen utilizarse algoritmos de entrenamiento o conjuntos de reglas locales cuyo objetivo es determinar los estados de equilibrio internos de las celdas para una serie de entradas y salidas dadas.

En un esquema tradicional, como el mostrado en la figura 1.3, las celdas de una red CNN pueden clasificarse en función de la posición que ocupan dentro de la red. Se denomina celda regular a aquella que está completamente rodeada por otras o que se encuentra completamente interconectada. Por el contrario, si la celda está parcialmente conectada o no está completamente rodeada se denomina celda frontera. Un caso particular de celda frontera, denomina celda esquina, es cuando la celda está situada en algunos de los vértices de la red.

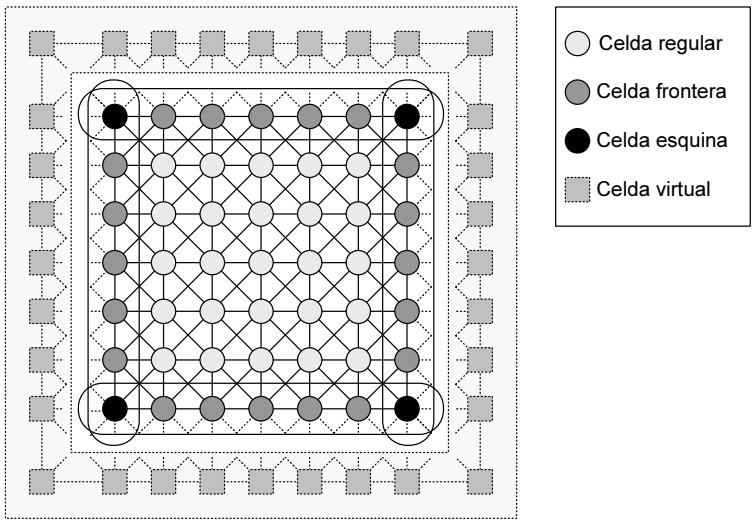


Figura 1.3: Clasificación de las celdas de una red CNN en función de su posición. Ejemplo para el caso de una red de tamaño 7×7 y radio de vecindad $r = 1$.

Según la definición de la sección 1.1, una red CNN no quedará completamente definida hasta que no se especifiquen las condiciones de su contorno. Para ello es necesario desarrollar un mecanismo que proporcione información a las entradas

no conectadas de las celdas fronteras. Este mecanismo se construye a partir de celdas virtuales, no implementadas físicamente, cuya única misión es recrear las condiciones de contorno de la red CNN. Es importante resaltar la importancia de las condiciones de contorno de las redes CNN debido a que, en virtud del efecto de propagación de la información, su valor podrá afectar de manera importante al comportamiento global de la red (P., 1993; Mirzai B., 1998). Las condiciones de contorno más utilizadas por la redes CNN son las siguientes (ver figura 1.4):

- **Condiciones de contorno fija (o contorno Dirichlet)**. Esta condición establece un valor fijo (habitualmente a cero) para las entradas de las celdas frontera.

- **Condiciones de contorno de flujo cero (o contorno Neumann)**. En este caso las celdas fronteras y las celdas virtuales comparten las mismas entradas y salidas.

- **Condiciones de contorno periódicas (o contorno toroidal)**. La red se pliega formando un cilindro, uniéndose las celdas frontera de posiciones simétricas, y a continuación los extremos del cilindro se unen para formar un anillo con forma toroidal.

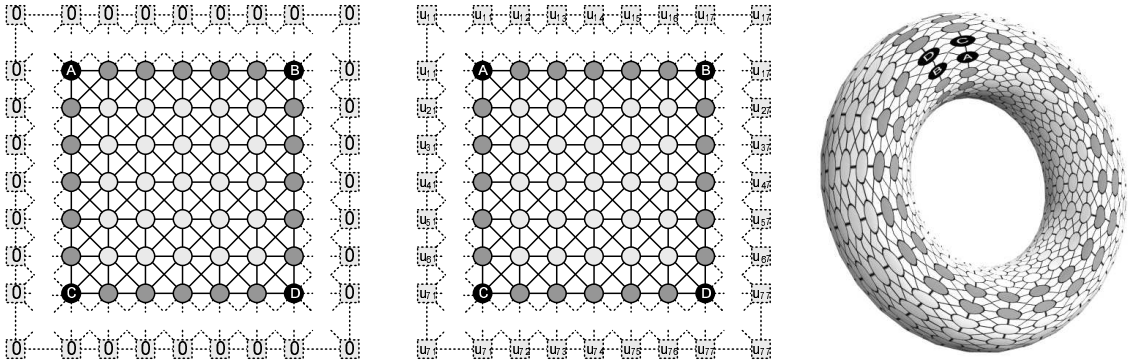


Figura 1.4: Condiciones de contorno típicas. a) Condiciones de contorno Dirichlet. b) Condiciones de contorno Neuman. c) Condiciones de contorno toroidal.

1.1.2. Celda estándar de la red CNN

Las celdas de una red CNN representan los nodos de interconexión de la estructura y constituyen las unidades básicas de procesamiento. Una red CNN puede estar formada por celdas idénticas o puede utilizar diferentes tipos, como ocurre en los sistemas biológicos reales. Internamente, las celdas son definidas a partir de modelos matemáticos, generalmente basados en ecuaciones diferenciales de bajo orden, que pueden incluir tanto elementos lineales como no lineales. Atendiendo a la cantidad de elementos utilizados y su capacidad de procesamiento, las celdas pueden ser simples o complejas. Un importante factor a tener en cuenta es que la evolución dinámica de las redes no dependerá tanto de la complejidad de las celdas como de la forma en la que fluye la información a su través.

En la arquitectura estándar de las redes CNN, las celdas están constituidas por los siguientes elementos (figura 1.5):

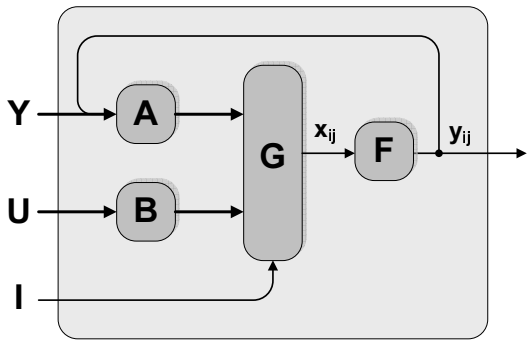


Figura 1.5: Diagrama de bloques de una celda estándar.

- Un conjuntos de entradas externas U, utilizadas para introducir la información externa que procesará la celda.

- Un conjunto de entradas de realimentación Y, por donde se reciben las salidas de las celdas vecinas, incluida la de la propia celda, que serán procesadas.

- Un operador sináptico B, que representa la matriz de pesos que pondera las correspondientes entradas U de la celda. La dimensión de esta matriz cuadrada para un radio de vecindad r dado es: $(2r + 1) \times (2r + 1)$.

- Un operador sináptico A, que representa la matriz de pesos que pondera las correspondientes entradas Y de la celda. La dimensión de esta matriz cuadrada para un radio de vecindad r dado es: $(2r+1) \times (2r+1)$.
- Un elemento de polarización (o bias) I, cuya misión es acelerar el aprendizaje y estabilizar la dinámica de la celda.
- Un elemento G, que rige la dinámica evolutiva de la celda.
- Una función de activación F, que se utiliza para transformar el valor de la variable de estado x_{ij}. Su respuesta representa la salida y_{ij} de la celda.

La función de activación F suele ser no lineal y se utiliza para acotar e interpretar la salida de la celda. En general, aunque es posible utilizar cualquier función matemática, lo habitual es usar alguna de las mostradas en la figura 1.6. De entre todas las funciones de activación, la más utiliza en la práctica, propuesta en el artículo original (Chua L., 1988), es la denominada función de activación estándar (figura 1.6-a). Esta función, conocida también como función cuasilineal, presenta ganancia unitaria en el centro y saturación en los extremos.

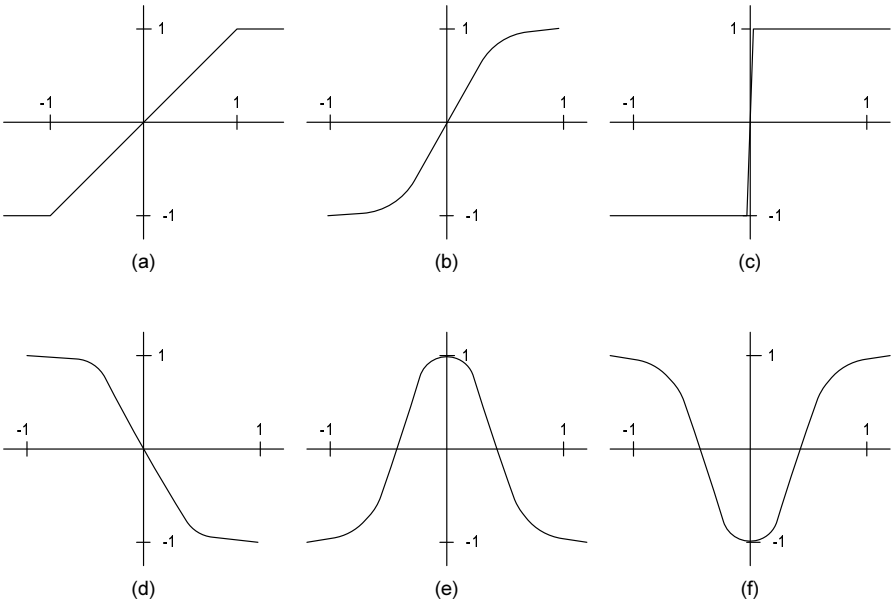


Figura 1.6: Funciones de activación típicas. a) Ganancia unitaria con saturación. b) Sigmoidal. c) Gran ganancia con saturación. d) Sigmoidal inversa. e) Gaussiana. d) Gaussiana invertida.

Los operadores A, B y el elemento I constituyen la plantilla de la celda. Dicha plantilla junto al elemento G determinan completamente el comportamiento de la red, para un conjunto de entradas dado y unas condiciones de contorno iniciales. La evolución dinámica de la celda es definida mediante el elemento G que, en la arquitectura estándar, queda representa mediante la siguiente ecuación de estado de primer orden (ecuación 1.2).

$$\dot{x}_{ij} = -x_{ij} + f(x_{ij}, U, Y, I) \tag{1.2}$$

El tamaño de los operadores A y B depende del radio de vecindad de las celdas y sus elementos, los pesos, determinarán la influencia de todas las entradas que recibe la celda. Los pesos de la plantilla podrán ser excitadores o inhibidores; si son excitadores, la variable de estado (x_{ij}) crecerá frente a entradas positivas; si son inhibidores, la variable de estado disminuirá frente a los mismos estímulos de entrada.

Una red CNN estándar se denomina clonada cuando las plantillas utilizadas por las celdas son invariantes a traslaciones espaciales, es decir, cuando todas las celdas utilizan la misma plantilla. En las redes CNN clonadas, la sustitución de una plantilla por otra permite obtener comportamientos completamente diferentes frente a las mismas entradas y condiciones de contorno. La gran cantidad de valores que puede tomar la plantilla, posibilita que una misma estructura pueda ser utilizada en multitud de aplicaciones. No obstante, en ocasiones, el uso de plantillas variables en el espacio o en el tiempo será beneficioso ya que permitirá obtener procesamientos más complejos y con menor coste computacional.

1.2. Variantes al modelo original de las redes CNN

En las secciones anteriores la arquitectura estándar de las redes CNN ha sido presentada mostrando sus principales características y posibilidades de configuración, fundamentalmente en función de su topología, conexionado y plantillas. A continuación se introducen las principales variantes al modelo original de la red CNN que ofrecen diferentes mejoras en cuanto a eficiencia, adaptación a nuevas aplicaciones y simplificación en su implementación. Todas estas variantes, resultantes de la adaptación del modelo tradicional al ámbito de la ingeniería electrónica, siguen manteniendo el espíritu de las redes CNN tradicionales:

- Cada celda cuenta con una dinámica interna propia.
- La conexión entre celdas es de carácter local y recurrente.
- Se utiliza un conjunto reducido de operaciones para controlar el comportamiento global de la red.

1.2.1. Redes CNN continuas en el tiempo

El modelo original de las redes CNN, conocido también como modelo de Chua y Yang, modelo estándar o modelo continuo de la red CNN (CT-CNN, *Continuous-Time* CNN), se caracteriza por tener un comportamiento continuo en el tiempo y una alta capacidad de procesamiento paralelo. Esta combinación de características permite que dicho modelo pueda ser utilizado en problemas específicos, que difícilmente pueden ser resueltos mediante otros modelos. Concretamente, las redes CT-CNN son utilizadas, por ejemplo, para estudiar los efectos caóticos que se producen en determinados sistemas no lineales complejos, como algunos tipos de circuitos resonantes (Arena P., 1998).

Las celdas de las redes CT-CNN son definidas mediante una ecuación diferencial lineal de primer orden (ecuación 1.3) y una función de activación, que habitualmente coincide la función estándar (ecuación 1.4, figura 1.6-a). Estas expresiones son tomadas como referencia para desarrollar las variantes del modelo más utilizadas (Cimagalli V., 1993).

$$\frac{d}{dt}x_{ij}(t) = -x_{ij}(t) + \sum_{k,l \in Nr(ij)} A_{kl}y_{kl}(t) + \sum_{k,l \in Nr(ij)} B_{kl}u_{kl}(t) + I_{ij} \tag{1.3}$$

$$y_{ij}(t) = f(x_{ij}(t)) = \frac{1}{2}(|x_{ij}(t)+1| - |x_{ij}(t)-1|) \tag{1.4}$$

1.2.2. Redes CNN con plantillas clonadas

El modelo estándar de las redes CNN deja abierta la posibilidad de utilizar plantillas diferentes en cada una de las celdas que constituyen la red. Dicha posibilidad, sin embargo, resulta difícil de considerar en una realización hardware

completa debido a los inconvenientes que supone el diseño y la implementación individual de cada una de las celdas. En general, para evitar este tipo de problemas, las redes CNN suelen utilizar la misma plantilla en todas sus celdas. El término empleado para denominar a este tipo de red, y que enfatiza la propiedad de invarianza de las plantillas, es el de redes CNN de plantillas clonadas (LCN-CNN, *Linear Cloning Template Celluar Neural Nerworks*).

En estas redes, la dinámica del sistema queda definida completamente por un conjunto de parámetros reducido, cuyo número depende únicamente del radio de vecindad de la celda. Si el radio de vecindad es 1 (ventana 3×3), la evolución del sistema vendrá determinada por una plantilla de tan sólo 19 parámetros, 9 por cada operador (A y B) y el correspondiente al bias (I). En general, para un radio de vecindad r, el número de parámetros requeridos por una red clonada será igual a $2(2r+1)^2 + 1$.

Este tipo de redes es utilizado en aplicaciones que requieren el mismo comportamiento de la red en todas las zonas del espacio. Este es el caso, por ejemplo, de las aplicaciones de procesamiento de imágenes que demandan el mismo efecto en toda la imagen (detección de contornos, umbralización, difusión, etc.). La gran versatilidad y sencillez de este tipo de redes hace que sean las más utilizadas y sencillas de implementar. Su amplio desarrollo ha dado lugar a una gran cantidad de librerías con multitud de plantillas diferentes.

1.2.3. Redes CNN de rango completo acotado

Un inconveniente que dificulta la realización hardware de las redes CNN es el alto valor que pueden alcanzar algunas de sus señales internas, en particular la variable de estado del sistema (x_{ij}). Esta señal, generalmente asociada a un nivel de tensión, puede alcanzar valores perjudiciales para el circuito o difícilmente suministrables por la fuente de alimentación. Para evitarlo, Rodríguez-Vázquez et al. (Rodriguez-Vazquez A., 1993) propusieron una modificación al modelo estándar de la red CNN, consistente en trasladar los límites de la función no lineal de salida a la variable de estado de la celda (figura 1.7). De esta forma, el rango de la variable de estado queda confinado en límites seguros, idénticos a los de las entradas y salidas, lo que supone un beneficio para la realización hardware sin perjuicio significativo para el comportamiento del modelo.

Las ecuaciones de estado de este tipo de redes, denominadas FSR-CNN (*Full Signal Range Celluar Neural Nerworks*), vienen representadas por las expresiones 1.5, 1.6 y 1.7, donde m representa una constante de diseño.

$$\frac{d}{dt}x_{ij}(t) = -g(x_{ij}(t)) + \sum_{k,l \in Nr(ij)} A_{kl}y_{kl}(t) + \sum_{k,l \in Nr(ij)} B_{kl}u_{kl}(t) + I_{ij} \tag{1.5}$$

$$y_{ij} = f(x_{ij}) = \frac{1}{2}(|x_{ij}+1| - |x_{ij}-1|) \tag{1.6}$$

$$g\left(x_{ij}(t)\right) = \begin{cases} m\left(x_{ij}(t)-1\right)+1 & si \quad x_{ij}(t) < -1 \\ x_{ij}(t) & si \quad |x_{ij}(t)| \leq 1 \\ m\left(x_{ij}(t)+1\right)-1 & si \quad x_{ij}(t) > 1 \end{cases} \tag{1.7}$$

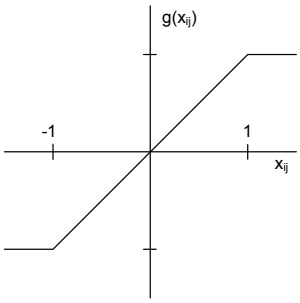


Figura 1.7: Función $g(x_{ij})$ utilizada por el modelo FSR-CNN.

1.2.4. Redes CNN no lineales

El modelo original de las redes CNN es considerado, generalmente, un sistema no lineal debido fundamentalmente a la utilización de una función de salida no lineal (ecuación 1.4). Sin embargo, dado que la ley dinámica que rige el comportamiento de la variable de estado (ecuación 1.3) si es lineal, algunos autores mantienen que el modelo en su conjunto debe ser considerado como un sistema lineal.

De modo indiscutible, una red CNN es no-lineal (NL-CNN, *Non-Lineal Cellular Neural Networks*) cuando el efecto de las entradas sobre las variables de estado responda de forma no lineal frente a estímulos que sí lo son, es decir,

cuando los elementos de las plantillas no son lineales. Las redes CNN con este comportamiento ofrecen una mejor aproximación a las redes neuronales biológicas y proporcionan una manera más eficiente y sofisticada de procesar la información (Chua L., 1993; Roska T., 1992). En general, este tipo de redes son necesarias para resolver problemas de clasificación cuya solución no es separable linealmente.

Se puede decir que las redes CNN no lineales han existido desde el principio (Roska T., 1992), ya que los componentes electrónicos utilizados en las primeras implementaciones analógicas (fuentes de corriente, resistencias, etc.) no ofrecían un comportamiento realmente lineal sino más bien una aproximación en torno a un punto de trabajo.

La no linealidad de las redes CNN puede darse tanto en estructuras simples como en estructuras formadas por múltiples capas. La forma más habitual de incorporar este comportamiento al modelo de la red CNN es añadir los términos de la expresión 1.8 a la ecuación de estado general del modelo estándar. Donde $\widehat{A}$ y $\widehat{B}$ son los operadores no lineales que se incorporan a la plantilla de las celdas.

$$\sum_{k,l \in Nr(ij)} \widehat{A}_{kl} y_{kl}(t) + \sum_{k,l \in Nr(ij)} \widehat{B}_{kl} u_{kl}(t) \tag{1.8}$$

1.2.5. Redes CNN retrasadas en el tiempo

Es sabido que los estímulos aferentes que recorren las redes neuronales biológicas pueden seguir diferentes trayectorias antes de llegar a su destino. Durante este recorrido, la información atraviesa diferentes elementos (interneuronas) que retrasan la información e impiden que los estímulos sensoriales alcancen su destino al mismo tiempo.

Este comportamiento temporal, patente en las redes neuronales biológicas, es el origen de las redes CNN retrasadas en el tiempo (D-CNN, *Delayed Cellular Neural Nerworks*). El principal objetivo que se persigue con este tipo de redes es que las celdas puedan trabajar con los mismos valores de las entradas en diferentes instantes de tiempo, lo que permite que puedan ser utilizadas en aplicaciones de análisis temporal como, por ejemplo, la detección de movimiento (Cimagalli V., 1990; Roska T., 1990).

El modelo D-CNN introduce un retraso ideal en las entradas de las celdas incorporando nuevos términos (ecuación 1.9) a la ecuación de estado original. Dichos términos tienen el mismo formato y dimensión que los operadores tradicionales (A y B) de la plantilla estándar.

$$\sum_{k,l \in Nr(ij)} A^{\tau}_{kl} y_{kl}(t-\tau) + \sum_{k,l \in Nr(ij)} B^{\tau}_{kl} u_{kl}(t-\tau) \tag{1.9}$$

En lo que respecta a su implementación, dado que el retraso ideal propuesto por el modelo es difícil de conseguir en hardware, lo que se suele hacer es aproximar dicho retraso mediante filtros paso-bajo de primer o segundo orden. Esta simplificación no supone ningún inconveniente en la funcionalidad, puesto que incluso los modelos biológicos más realistas consideran que los retrasos en el sistemas nervioso no son ideales, sino que presentan respuestas equivalentes a sistema diferencial de bajo orden (Chua L.O, 2002).

1.3. Redes CNN discretas en el tiempo

Uno de los modelos de red CNN que más interés ha despertado desde su aparición ha sido el de las redes CNN discretas en el tiempo (DT-CNN, *Discrete-Time Cellular Neural Networks*). Concebidas (Harrer y Nossek 1992) como una versión discreta de las redes CNN continuas, las DT-CNN fueron definidas originalmente para hacer uso de una función de salida tipo umbral (figura 1.6-c), lo que, por otra parte, no significa que el modelo tenga que utilizar siempre esta función.

En la definición general del modelo de las DT-CNN, dada por Chua y Roska (Chua L.O, 2002), se establece que la salida de este tipo de redes podrá ser cualquier función de activación, continua o discontinua. Para el caso de aplicaciones de procesamiento de imágenes en escala de grises, las funciones de salidas más habituales son la función de activación estándar y la función de umbral original, en este último caso, las redes pasan a denominarse DT-CNN *hard limiter*.

La obtención del modelo de las redes CNN discretas se consigue sustituyendo, en la ecuación de estado continua (ecuación 1.3), la variable temporal t por una aproximación discreta en el tiempo $t = nh$, siendo h la constante de muestreo y n la variable que define el número de iteraciones que ejecuta el modelo. A partir

de esta sustitución se obtiene la expresión 1.10

$$\frac{1}{h}\left[x_{ij}\left((n+1)h\right) - x_{ij}(nh)\right] = -x_{ij}(nh) + \sum_{k,l\in Nr(ij)} A_{kl}y_{kl}(nh) + \sum_{k,l\in Nr(ij)} B_{kl}u_{kl}(nh) + I_{ij} \tag{1.10}$$

Reordenando términos y sustituyendo las variable continuas x_{ij} e yij por sus versiones discretas (ej. $x_{ij}(nh)$ por $X_{ij}[k]$) se llega a la ecuación 1.11

$$X_{ij}[k+1] = (1-h)X_{ij}[k] + h\left(\sum_{k,l\in Nr(ij)} A_{kl}Y_{kl}[k] + \sum_{k,l\in Nr(ij)} B_{kl}U_{kl}[k] + I_{ij}\right) \tag{1.11}$$

Haciendo $h = 1$, sin pérdida de generalidad, y utilizando la función de activación umbral se obtienen las expresiones en diferencias que definen el modelo original de la DT-CNN *hard limiter*.

$$X_{ij}(k+1) = \sum_{k,l\in Nr(ij)} A_{kl}f\left(X_{kl}(k)\right) + \sum_{k,l\in Nr(ij)} B_{kl}Y_{kl}(k) + I_{ij} \tag{1.12}$$

$$Y_{ij}(k) = f\left(X_{ij}(k)\right) = \begin{cases} 1 & \text{si} \quad X_{ij}(k) > 0 \\ -1 & \text{si} \quad X_{ij}(k) < 0 \end{cases} \tag{1.13}$$

Hay que indicar que, en el modelo original, la función $f\left(X_{ij}(k)\right)$ no fue definida para el caso particular $X_{ij}(k) = 0$. No obstante, para una implementación analógica, para la cual fue concebido inicialmente el modelo, esto no supone ningún inconveniente ya que el ruido interno del circuito evita la indeterminación de esta variable. En implementaciones digitales este aspecto sí deberá ser considerado y será necesario especificar un valor concreto de la salida (+1 o -1) en dicho punto.

Las DT-CNN han sido utilizadas en multitud de aplicaciones de procesamiento de imágenes, usando diferentes funciones de activación, plantillas clonadas o no lineales (Vilariño et al., 1998; Aomori et al., 2005; Hsin C., 2006; Harrer 1993). Algunas de las ventajas más interesantes de las DT-CNN se resumen a continuación:

- Son fácilmente simulables en un computador, dado que no necesitan algoritmo de integración.

- Su implementación como circuito integrado es más robusta que la implementación analógica, ya que son mucho menos sensibles al ruido y a las variaciones de los parámetros específicos del proceso de fabricación.

1.4. Redes CNN multicapa

Los modelos de las redes CNN consideradas en las secciones anteriores tienen como denominador común que son sistemas donde cada una de sus celdas contribuye con una única variable de estado a la evolución dinámica del conjunto. Dichos sistemas permiten resolver multitud de tareas de procesamiento, aunque en general, no disponen del potencial de cálculo suficiente para resolver problemas de alta complejidad (Vilariño, 2001). Una opción que puede ayudar en estos problemas consiste en dividir las tareas más complejas en subtareas más sencillas y abordar su solución a partir de sucesivas redes CNN de una única capa, es decir, mediante redes CNN multicapa.

Las redes CNN multicapa (ML-CNN, *Multi Layer Cellular Neural Networks*) permiten generalizar un modelo de red CNN, en el que se considera un mayor orden del sistema o un incremento en el número de variables de estado de las celdas. De esta manera, una red ML-CNN podrá estar compuesta por una única capa de celdas con varias variables de estado, o por un conjunto de varias capas formadas por celdas de una sola variable de estado. Esta última estructura, originalmente pensada para ser implementadas mediante redes DT-CNN, puede ser imaginada como una composición de varias capas simples, apiladas o superpuestas, entre las que existe una interacción directa. Cada capa es diseñada para llevar a cabo un procesamiento determinado e interaccionar en paralelo con las demás. El carácter síncrono de las redes DT-CNN facilita el desarrollo de sistemas modulares basados en la sucesión de capas simples. Las estructuras elementales mostradas en la figuras 1.8, propuestas en (Harrer 1993), permitirían construir redes ML-CNN de alta complejidad.

La conexión paralela (figura 1.8-a), combina las salidas de las capas l y m mediante una función lógica f_L del tipo AND, XOR, etc. En el modo de conexión serie (figura 1.8-b), la salida de cada capa es conectada a la entrada de la siguiente capa. El modo de conexión en realimentación (figura 1.8-c) es similar al modo

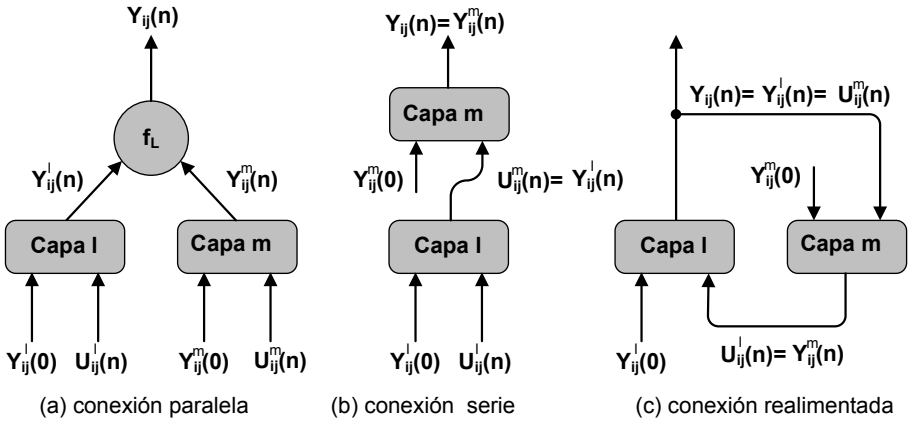


Figura 1.8: Modos elementales de interconexión de las capas en una red ML-CNN.

serie, salvo que la salida de la capa m es conectada a la entrada de la capa l para cerrar el lazo de realimentación.

Aunque en el artículo original (Harrer 1993) no se contempla la conexión de la salida Y_{ij} de un módulo con la entrada Y_{ij} de otro, esta interconexión es utilizada por algunos autores (Johan et al., 1996). Dado que no existe ninguna razón para lo contrario, los modos de conexión elementales de la figura anterior pueden ser extendidos con los mostrados en la figuras 1.9. La combinación de todos los modos de interconexión permitirá desarrollar cualquier estructura ML-CNN.

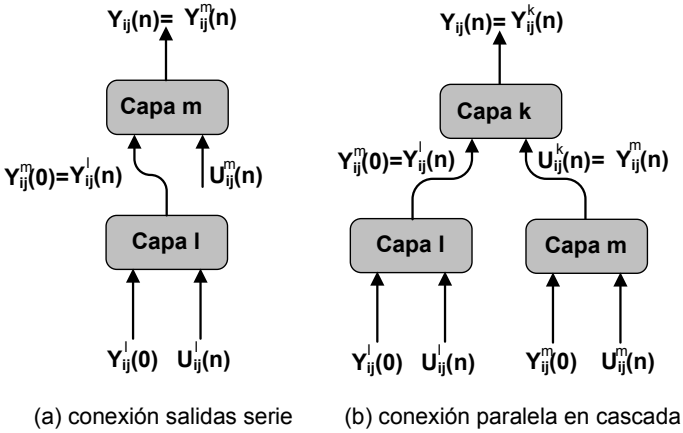


Figura 1.9: Reglas adicionales para la interconexión de capas en redes ML-CNN.

Es interesante destacar que el hardware extra que requieren las redes ML-CNN puede ser reducido mediante la utilización de estructuras monocapa realimentadas con plantillas reprogramables, es decir, variables en el tiempo. Esta

estructura, que puede estar formada por únicamente dos capas, como muestra la figura 1.8-c, podría llevar a cabo las mismas operaciones que realizan múltiples capas conectadas en serie.

1.5. Redes CNN universal machine

Como se ha indicado en la sección anterior, en ocasiones, las operaciones básicas realizadas por las redes CNN de una sola capa no son suficientes para resolver problemas de alta complejidad, por lo que resulta necesario recurrir al uso de redes CNN de más de una capa. En otros casos, independientemente de la complejidad del problema, la aplicación de múltiples redes CNN de una sola capa es necesaria para combinar diferentes procesamientos y conseguir la respuesta deseada. En todos estos casos, en general, las capas utilizarán distintas plantillas y condiciones de contorno iniciales. Aunque la realización de estos sistemas puede llevarse a cabo mediante CNN-ML, la dificultad que entraña su implementación hardware, inicialmente pensada para circuitos integrados de aplicación específica (ASIC, *Application Specific Integrated Circuit*) y basada en estructuras rígidas con plantillas fijas, dio lugar al desarrollo de las redes CNN programables.

Las redes CNN programables representaron un paso adelante en el desarrollo práctico del paradigma de las redes CNN multicapa. Las redes CNN programables, tradicionalmente denominadas redes CNN *Universal Machine* (CNN-UM) (Roska y Chua 1993; Roska y Rodriguez 2000), fueron concebidas como matrices de procesadores programables donde los elementos estructurales, las celdas, constituyen las unidades de procesamiento y las plantillas sus instrucciones.

El término "*Universal*", con el que se acuña a este tipo de redes, hace referencia a su capacidad para funcionar como máquinas de Turing. En este sentido, se ha demostrado teóricamente (Kenneth y Chua 1996) que tanto las redes CNN-UM como las redes CNN en general (Chua et al., 1993) pueden ser utilizadas para resolver cualquier tipo de problema computacional. No obstante dicha demostración, por desgracia, no da respuesta a la definición de los algoritmos necesarios para resolver los problemas en cuestión. De hecho, en la actualidad, una de las principales líneas de trabajo abiertas en el desarrollo de las redes CNN, a la espera de solución, es el aprendizaje y el diseño de plantillas.

Las operaciones realizadas por las redes CNN-UM pueden ser interpretadas como la ejecución de diferentes instrucciones sobre un computador, es decir, los datos son procesados por una plantilla y, a continuación, el resultado es almacenado en una memoria y procesado por otra plantilla diferente. Las CNN-UM han sido desarrolladas ampliamente y disponen de gran cantidad de librerías (Yang 2002; Manganaro, 1999; Chua L.O, 2002; Roska et al., 2000) de plantillas que realizan tareas de procesamiento básicas. La concatenación de varias de estas tareas puede dar lugar a complejos algoritmos de procesamiento.

Las redes CNN-UM fueron diseñadas para ser implementadas sobre dispositivos ASIC y ampliar las capacidades de las redes CNN convencionales, añadiendo nuevas funcionalidades a las celdas e introduciendo nuevos bloques hardware analógicos y digitales. Estos nuevos bloques, constituidos principalmente por elementos de memoria, almacenan tanto los datos del procesamiento como la información de configuración de las celdas y de sus plantillas. Las celdas de la CNN-UM incluyen también cierta lógica de control que sincroniza la arquitectura y hace que su ejecución sea más eficiente. La mezcla de partes digitales y analógicas dentro del mismo chip, sin intervención de ningún tipo de convertidor analógico-digital o digital-analógico, dio lugar a un término nuevo denominado computación dual o computación *analogic* (de la contracción de las palabras inglesas *analog* y *logic*).

A partir de este concepto se han desarrollado diferentes tipos de chips que utilizan la arquitectura programable de la CNN-UM para llevar a cabo multitud de tareas de procesamiento. En estos sistemas, la proyección de los algoritmos de procesamiento sobre el hardware se realiza mediante plataformas software que incluyen librerías, simuladores, compiladores y hasta su propio sistema operativo.

En la figura 1.10 se muestran las diferentes etapas que se siguen para trasladar el algoritmo de procesamiento a la plataforma hardware de la CNN-UM (Roska y Rodriguez 2000). En primer lugar, el algoritmo que define la aplicación debe ser descrito mediante un lenguaje de alto nivel que se denomina alpha. Posteriormente, utilizando un compilador específico, el código es traducido a otro código de menor nivel, llamado AMC *Analogic Macro Code*, que puede ser ejecutado sobre tres plataformas diferentes: un simulados basado en PC, una placa de desarrollo hardware denominada *CNN-Chip-Prototyping-System* (CCPS), y un emulador digital basado en dispositivos DSP (*Digital Signal Processor*). La principal ventaja de este sistema radica en la posibilidad de poder comprobar el funcionamiento de

los algoritmos mediante simulación, antes de la programación del chip.

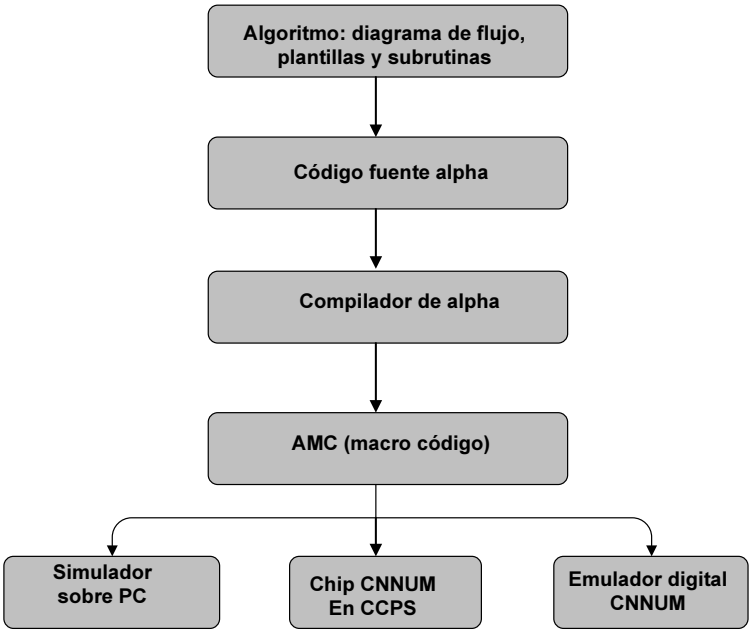


Figura 1.10: Etapas software asociadas al desarrollo de aplicaciones con CNN-UM.

1.6. Simulación y emulación digital de las redes CNN.

La implementación de las redes CNN sobre circuitos ASIC analógicos es la solución más eficiente cuando se desean obtener las máximas prestaciones en la velocidad de procesamiento con el menor coste en área de silicio. Sin embargo, la dificultad para acceder a estos chips, su considerable complejidad, su tiempo de desarrollo y alto coste de fabricación, hace necesario buscar de otras alternativas de implementación que permitan simplificar el desarrollo de las aplicaciones y el diseño de plantillas.

Las herramientas de simulación software, como "Sirena" (Dominguez et al., 1994) o "SimCNN" incluidas en el paquete CADETwin (*CNN Application Development Enviroment and Toolkit for Windows NT*) (Roska et al., 1997; Szolgay et al., 1999), han sido utilizadas con gran éxito en el estudio y desarrollo de aplicaciones basadas en redes CNN. No obstante, la simulación software tiene el

inconveniente de que para aplicaciones de gran tamaño las operaciones realizadas por la redes CNN no pueden ser calculadas en tiempo real. Esto, junto a los inconvenientes asociados con la implementación ASIC y la baja precisión de las redes CNN analógicas (Nagy y Szolgay 2004), fueron las razones principales que dieron lugar a la aparición de sistemas digitales encargados de emular a las redes CNN. Dichos sistemas, tienen la ventaja de que son más sencillos de implementar que los sistemas analógicos y, aunque presentan limitaciones en cuanto a velocidad, su velocidad de procesamiento es mayor que la de los sistemas de simulación software, por lo que tienen mayor oportunidad de ser utilizados en aplicaciones de tiempo real.

El primer sistema digital utilizado para emular las redes CNN fue denominado *CNN-HAB* (*CNN Hardware Accelerator Board*) (Roska et al., 1992). Esta plataforma, basada en múltiples procesadores de tipo DSP, fue desarrollada para conseguir mayores velocidades de procesamiento que las obtenidas por las herramientas de simulación software anteriores. El éxito obtenido por esta plataforma fue la razón que motivó el desarrollo de nuevos circuitos ASIC digitales dedicados específicamente a resolver la ecuación de estado del modelo de la red CNN. La especialización de estos circuitos en dichos cálculos, permitió disminuir los requerimientos de área de los circuitos y posibilitó la implementación de múltiples elementos de procesamiento en un único circuito integrado. Como consecuencia, el número de componentes de los sistemas de desarrollo se redujo considerablemente (número de chips), lo que minimizó a su vez los requerimientos de espacio y consumo de los sistemas.

La utilización de circuitos ASIC digitales supuso un gran paso adelante en la emulación de las redes CNN. Sin embargo, como sucede con los ASIC analógicos, aunque en menor escala, el tiempo de desarrollo y el coste de fabricación continúan siendo un inconveniente para la realización. Una alternativa que mejora enormemente la flexibilidad del diseño y que reduce considerablemente el coste y el tiempo de desarrollo de los circuitos digitales, es el empleo de dispositivos hardware reconfigurables (FPGA). Esta tecnología permite implementar cualquier función digital de forma rápida y sencilla, y además ofrece la ventaja de poder realizar modificaciones y actualizaciones cuando se desee, sin coste de fabricación adicional.

Capítulo 2

Soluciones para la realización de las redes CNN

Las redes CNN son un tipo particular de Redes Neuronales Artificiales (ANN) creadas específicamente para ser implementadas en hardware y utilizadas, principalmente, en aplicaciones embebidas de procesamiento de imágenes en tiempo real. La primera implementación hardware de la red CNN como circuito integrado fue realizada a principios de los años 90 (Harrer y Nossek 1992), a partir del modelo discreto, usando componentes electrónicos analógicos. Desde este primera realización, la aparición de nuevos chips, cada vez más sofisticados y con mayores prestaciones, ha sido constante como consecuencia de los avances en la tecnología de fabricación de los circuitos y la mejora de las técnicas y herramientas de diseño electrónico.

A lo largo de la evolución en la realización de las redes CNN On-Chip destacan contribuciones de gran interés como la primera implementación de las redes DT-CNN (Harrer et al., 1993). Otras contribuciones relevantes, que han tratado de resolver los inconvenientes de la interfaz de entrada y salida de las realizaciones analógicas, han sido las implementaciones ópticas de las redes CNN basadas tanto en modelos continuos (Sullivan et al., 1996) como en modelos discretos (Buczynski et al., 1998). En la última década, la tendencia con mayor impulso ha sido la emulación digital, principalmente motivada por el auge de los dispositivos lógicos reconfigurables y las GPU.

En este capítulo abordaremos el estudio de las diferentes soluciones utilizadas para la realización de sistemas basados en redes CNN, poniendo especial interés en las soluciones hardware de propósito específico, dado que ofrecen mejores prestaciones para el desarrollo de sistemas de procesamiento en tiempo real. A continuación se realizará un repaso de las aportaciones más importantes realizadas en este ámbito.

2.1. Realización de las redes CNN

Las redes CNN puede ser realizadas usando tanto procedimientos software como técnicas de diseño hardware específicas: analógicas, digitales o mixtas. La alternativa utilizada influirá determinantemente en las características del sistema a desarrollar (complejidad, eficiencia, precisión, etc.) y, por consiguiente, su elección deberá ser consecuente con los requisitos y necesidades de cada aplicación.

La realización de las redes CNN puede llevarse a cabo utilizando principalmente tres técnicas distintas: simulación software, implementación analógica y emulación digital. La implementación analógica implica en general, la realización completa de la red CNN, es decir de todas sus celdas. En este caso, se hace uso de componentes electrónicos analógicos y el diseño puede partir tanto del modelo continuo de la red CNN (CT-CNN) como del modelo discreto (DT-CNN). La emulación digital y la simulación software, sin embargo, siempre están basadas en modelos discretos de la red CNN por lo que serán ejecutadas sobre plataformas de cómputo digital. En el caso de la emulación digital, dado que el tamaño de los componentes digitales es mayor que el de los componentes analógicos, la implementación casi nunca es completa y, en general, únicamente se implementa parte de la red. Esto sin embargo no supondrá una pérdida de funcionalidad en el sistema, aunque sí, probablemente, una pérdida de prestaciones en cuanto a velocidad.

Dentro de las técnicas utilizadas para realizar las redes CNN existen multitud de variantes, con distinto grado de adecuación, que el diseñador deberá considerar con el fin de seleccionar aquella que mejor se adapte a las necesidades y restricciones de cada aplicación. Cuando varias soluciones garanticen dichas restricciones, el diseñador deberá elegir aquella que proporcione las mejores prestaciones. En es-

te sentido, los parámetros más importantes a tener en cuenta a la hora de medir y determinar la eficiencia de estas realizaciones son: la velocidad de procesamiento, la precisión en los resultados, la cantidad de recursos utilizados en la realización (hardware o software), el consumo de energía, la flexibilidad en el diseño, el espacio físico necesario (desde el enfoque de los sistemas embebidos) y, por último, el tiempo y el coste de desarrollo y de fabricación.

Cuando la alternativa elegida para la realización es la simulación software, bien porque sólo interesa analizar el funcionamiento, hacer un prototipo, o simplemente porque la aplicación no exige altas prestaciones en velocidad, las opciones disponibles son varias. Todas tienen en común el uso de un software de simulación y se diferencian principalmente en la plataforma de cómputo utilizada, la cual interesa que resuelva la dinámica de la red en el menor tiempo posible. Dado que el modelo de la red CNN está definido por un conjunto de ecuaciones diferenciales, la solución software deberá estar basada en un método de integración numérica, entre los cuales, algunos de los más utilizados son: Euler, Trapecio, Runge-Kutta, Gauss, Simpson, etc. Hay que indicar, sin embargo, que el comportamiento de las redes CNN depende en mayor medida de como se propaga la información a lo largo de la red que del método de integración utilizado. De hecho en muchas aplicaciones, la elección de un método u otro no produce cambios significativos en los resultados, y por tanto, el método más usado suele ser el más sencillo.

El rendimiento de la simulación software de las redes CNN depende de las prestaciones del sistema de cómputo y en particular de su capacidad para ejecutar la mayor cantidad de iteraciones por unidad de tiempo. En este sentido, las redes CNN pueden ser simuladas usando distintas plataformas de cómputo. Las plataformas secuenciales, constituidas por uno o varios elementos de procesamiento como, por ejemplo, un PC, utilizan lenguajes de programación convencionales: C, Java, Matlab..., y suelen ser una solución, relativamente, lenta para llevar a cabo la simulación. Cuando se desea mayor velocidad, una posibilidad es utilizar sistemas de cómputo de mayores prestaciones (HPC, High Performance Computer), compuestos por múltiples procesadores, que hacen uso de lenguajes de programación paralela para obtener las máximas prestaciones del sistema. Otra alternativa que cada vez está tomando mayor interés es la simulación de las redes CNN mediante unidades de procesamiento gráfico (GPU, Graphics Processing Units). Estos dispositivos, constituidos por varios núcleos de procesamiento en un sólo

chip, disponen de elementos de procesamiento altamente segmentados diseñados, específicamente, para realizar operaciones vectoriales orientadas a gráficos.

Cuando la solución software no es capaz de proporcionar las prestaciones requeridas por una aplicación, principalmente porque es necesario mayor velocidad de procesamiento o dicha aplicación debe ser ejecutada en tiempo real, las opciones hardware analógico o digital pueden ser la solución.

Cuando la red CNN es implementada mediante circuitos electrónicos analógicos, la eficiencia del sistema se incrementa y es posible conseguir las máximas prestaciones en cuanto a velocidad de procesamiento, consumo de energía y uso de área de silicio. Los circuitos analógicos, sin embargo, presentan importantes inconvenientes que limitan su utilización, como, por ejemplo, un mal comportamiento frente al ruido, alta sensibilidad a variaciones de temperatura y de tensión, alta dependencia a factores tecnológicos relacionados con el proceso de fabricación (parásitos, imperfecciones y tolerancias), etc. Como consecuencia, los cálculos realizados por los circuitos analógicos tienen menor precisión que los realizados por los circuitos digitales y su campo de aplicación se reduce. Además, la dependencia de los factores tecnológicos hace que las herramientas de diseño empleadas sean difíciles de automatizar y que, por tanto, sea necesario utilizar técnicas de diseño manual, que incrementan el tiempo de desarrollo de los circuitos y los costes de fabricación.

Otro inconveniente asociado al uso de circuitos integrados analógicos es su falta de flexibilidad frente a cambios en el diseño, lo que obliga a rediseñar gran parte del sistema frente a cualquier modificación del circuito. Este inconveniente, junto con la limitada capacidad de reprogramación de estos circuitos, hace que sea prácticamente imposible desarrollar estructuras CNN complejas basadas en plantillas no lineales, variables en el tiempo o dependientes. En la actualidad, los circuitos electrónicos analógicos son utilizados principalmente para desarrollar redes CNN de estructuras simples y de pequeño tamaño.

Los circuitos electrónicos digitales suelen ser usados para realizar la emulación hardware de las redes CNN, ya que presentan menor eficiencia que los circuitos analógicos en cuanto a velocidad de procesamiento, ocupación de área de silicio y consumo de energía. Los circuitos digitales, sin embargo, tienen como ventaja que son mucho más precisos y robustos que los circuitos analógicos. Además, el

proceso de diseño de estos circuitos es más flexible y sencillo que el de los circuitos analógicos, lo que facilita el desarrollo de redes CNN complejas compuestas por múltiples capas, plantillas variantes, no lineales, etc. La emulación digital de las redes CNN puede llevarse a cabo utilizando circuitos integrados de aplicación especifica (ASIC, *Application Specific Integrated Circuits*), o dispositivos hardware reconfigurables (como las FPGAs).

Las FPGAs permiten desarrollar circuitos digitales con prestaciones de velocidad intermedias entre las soluciones software y las alternativas hardware basadas en circuitos ASIC analógicos. La principal característica de estos dispositivos es su metodología de diseño, independiente de la tecnología de fabricación del chip, que tiene como principal ventaja la posibilidad de reconfiguración de los circuitos. Este aspecto incrementa la flexibilidad del diseño y permite que los circuitos implementados puedan ser modificados de forma rápida y sencilla en cualquier momento. El proceso de reconfiguración es extremadamente rápido y económico comparado con el proceso de fabricación de los circuitos ASIC.

2.2. Soluciones software para la simulación de las redes CNN

Las simulaciones de las redes CNN basadas en microprocesadores de propósito general, o sistemas equivalentes, hacen uso de plataformas de cómputo independientes de la aplicación para llevar a cabo la ejecución. Los beneficios de estas soluciones redundan en una mayor flexibilidad para realizar cambios en los algoritmos, reutilizar los sistemas en diferentes aplicaciones y conseguir procesamientos y estructuras CNN de mayor complejidad. Por otro lado, el hecho de que algunos de estos sistemas realicen un procesamiento puramente secuencial no beneficia al desarrollo de aplicaciones basadas en redes CNN, dado su marcado carácter paralelo.

En aplicaciones con pocos requisitos de velocidad, o cuando la flexibilidad en el diseño es prioritaria, la simulación software de la red CNN es la solución recomendable debido a su sencillez y facilidad de desarrollo. Las ventajas de esta alternativa permiten afrontar de forma cómoda y sencilla cambios en la topología y dimensiones de la red, así como en su estructura de capas y funcionalidad. En

general, las principales aplicaciones donde es utilizada la simulación software de las redes CNN son en el diseño de plantillas (Yin et al., 1999), el análisis de la estabilidad y dinámica de la red (Hanggi y Moschytz 1997) y el estudio de la viabilidad de determinadas aplicaciones (Perko et al., 1998).

Las herramientas de desarrollo más utilizados para llevar a cabo la simulación de las redes CNN, entre los que cabe destacar los simuladores (CNNsim, 2001; CELL, 2001; CANDY, 2001), han sido diseñadas para ser ejecutados sobre plataformas de cómputo convencional (ordenadores personales), lo que en general implica un bajo rendimiento computacional. Una alternativa para mejorar la velocidad consiste en ejecutar las simulaciones sobre plataformas con mayor capacidad de procesamiento.

Una de las principales características de las redes CNN es su capacidad para evaluar y calcular el estado de todas sus celdas de manera simultánea. Esto puede ser aprovechado mediante el uso de plataformas de cómputo paralelas como las estaciones de trabajo o, en general, los sistemas de cómputo de altas prestaciones (HPC, *High Performance Computer*). Estos sistemas, formados por numerosos procesadores o cluster de procesadores, cuentan con un alto nivel de paralelismo y lenguajes adaptados, que permiten acelerar, varios órdenes de magnitud, la simulación de las redes CNN (Schikuta y Weishäupl 2003). No obstante, hay que tener en cuenta que la utilización de estos sistemas exige un pequeño esfuerzo a la hora de adecuar los algoritmos de simulación (Weishupl y Schikuta 2004) a las características y requisitos de su paralelismo.

Otra alternativa que permite incrementar la velocidad de simulación de las redes CNN, un orden de magnitud con respecto a un PC convencional, es la utilización de unidades de procesamiento gráfico (GPU, Graphics Processing Units) (Tze-Yui et al., 2008). Estos dispositivos proporcionan un alto rendimiento computacional debido a que han sido diseñados específicamente para acelerar operaciones de tipo vectorial similares a las que realizan las redes CNN. Los dispositivos GPU incluyen en el mismo chip varios núcleos de procesamiento con arquitecturas altamente segmentadas y de gran eficiencia computacional. Algunos de los procesadores GPU utilizados para evaluar el rendimiento de la simulación de las redes CNN (Soos et al., 2008) sobre tarjetas gráficas comerciales son los dispositivos: G80GL, G80, G71 de Nvidia y el procesador Cell de la alianza entre Sony, Toshiba e IBM.

De entre todas las aplicaciones donde las redes CNN pueden ser utilizadas, las de procesamiento de imágenes y video en tiempo real suelen ser, en general, las más exigentes en cuanto a requerimientos de velocidad de procesamiento. En muchos casos, dichos requerimientos implican que la simulación software no puede ser utilizada para alcanzar las prestaciones mínimas exigidas. Otro tipo de aplicaciones donde tampoco es posible aprovechar las ventajas de las herramientas de simulación software son aquellas que presentan altas restricciones de espacio y de consumo, como, por ejemplo, el desarrollo de sistemas embebidos. Para este tipo de aplicaciones, las soluciones basadas en hardware dedicado pueden ser la alternativa más adecuada.

2.3. Soluciones hardware para la implementación de las redes CNN

Las redes CNN fueron desarrolladas inicialmente para ser implementadas en hardware utilizando circuitos integrados analógicos de propósito específico (ASIC, *Application Specific Integrated Circuits*). No obstante, esto no significa que su realización sobre dichos circuitos sea sencilla y que esté exenta de dificultades, todo lo contrario. En todas las fases y niveles de abstracción involucrados en el diseño y la realización de un circuitos electrónico son requeridos altos conocimiento técnico y gran capacidad de toma de decisiones para solucionar, de la manera más apropiada posible, las incertidumbres que surgen durante el desarrollo.

Las redes CNN cuentan con dos características fundamentales que permiten que su realización hardware sea viable.

- La utilización de un modelo de interconexión local entre celdas.
- Un conjunto de requerimientos computacionales lo suficientemente sencillo como para poder ser implementado en hardware.

Tradicionalmente, la implementación hardware de las redes CNN ha sido realizada a partir de dispositivos VLSI (*Very Large Scale Integration*), donde es posible integrar los elementos imprescindibles para el funcionamiento de las redes CNN. Con este tipo de circuitos es posible minimizar el tamaño de los sistemas

de desarrollo y mejorar las prestaciones funcionales y energéticas de las aplicaciones. La reducción de tamaño y la especialización conseguida permiten obtener grandes beneficios, aunque también generan importantes desventajas, entre las que podemos destacar la dificultad en el diseño y el tiempo de desarrollo. A continuación, se exponen las principales características, ventajas y desventajas de las tecnologías de diseño más utilizadas para la fabricación de circuitos integrados.

2.3.1. Diseño y realización de circuitos integrados

Para abordar la realización de un circuito electrónico integrado es preciso determinar a priori que técnica de diseño será conveniente utilizar: analógica, digital o mixta. Una vez decidido, será posible definir la estructura, la jerarquía y las especificaciones del circuito a diseñar. A continuación se emplearán diferentes metodologías de diseño para materializar físicamente el circuito, para lo cual, previamente, será necesario elegir la tecnológica de integración a utilizar. En el ámbito de las tecnológicas VLSI, existen dos alternativas que permiten desarrollar circuitos microelectrónicos de aplicación específica: las basadas en circuitos integrados de aplicación específica (ASIC) y las basadas en dispositivos lógicos reconfigurables (como las FPGAs).

La tecnología elegida, junto con la metodología y la técnica de diseño apropiada, determinarán las principales características y prestaciones de los circuitos implementados: velocidad, área de silicio ocupada, consumo de energía, coste de producción, etc. La elección de la tecnología y de las técnicas de desarrollo dependerá de las restricciones impuestas por la aplicación y del compromiso que se desee alcanzar en cuanto a versatilidad y prestaciones.

Cuando lo que interesa es obtener las máximas prestaciones de un circuito, la tecnología de fabricación ASIC es la que proporciona los mejores resultados. Con ella es posible desarrollar chips completamente a medida (*full-custom*), usando componentes e interconexiones que pueden ser optimizados al más bajo nivel. Para conseguirlo, el circuito es definido completamente a nivel físico, mediante procesos químicos y mascaras litográficas, con el fin de definir exactamente sus parámetros de fabricación: anchura del canal de los transistores, el tamaño de pistas, los niveles de metalización, emplazamiento de los componentes, etc. La técnica de diseño *full-custom* (figura 2.1-a) es aplicable únicamente a la tecnología ASIC y

es utilizada fundamentalmente para fabricar dispositivos analógicos y digitales con las más altas prestaciones (amplificadores, microprocesadores, memorias,...).

Entre las principales ventajas de los circuitos ASIC *full-custom* destacan su menor consumo de energía y área de silicio y una mayor velocidad de funcionamiento. No obstante, en ocasiones estas ventajas quedan eclipsadas por los inconvenientes que supone su producción, debido a su alta complejidad de desarrollo, tiempo de diseño y coste de fabricación. La gran inversión que es necesario realizar para fabricar un circuito *full-custom* hace que solo pueda ser amortizada cuando la producción del circuito es muy elevada o cuando las prestaciones del circuito no puedan ser obtenidas por otros medios. La técnica *full-custom* es la técnica de diseño más compleja debido a su gran dependencia de factores de tipo tecnológico, lo que provoca que las herramientas de diseño sean difíciles de automatizar y que, en la actualidad, los circuitos aún se sigan desarrollando de manera manual. Todo esto eleva el coste de producción e impide garantizar el funcionamiento de los circuitos antes de la fabricación.

Otra alternativa ASIC que también puede ser utilizada, es la técnica de diseño *semi-custom*. Esta solución ofrece ventajas, con respecto a la técnica *full-custom*, en términos de coste y tiempo de desarrollo, al introducir cierto grado de automatización en el diseño de los circuitos. La técnica *semi-custom* puede ser aplicada sobre diferentes tecnologías de fabricación: celdas estándar (*standard cell*), matrices de puertas (*gate array*) o el diseño estructurado (*structured array*).

La tecnología desarrollada a partir de celdas estándar (figura 2.1-b) se basa en el uso de bloques funcionales que previamente han sido diseñados, optimizados y caracterizado por el fabricante del ASIC. Estos bloques, denominados celdas estándar, proporcionan una funcionalidad básica a nivel de puerta o de transistor y representan las primitivas elementales del circuito. Los bloques se suministran en paquetes y bibliotecas de componentes, y el diseñador los combina, ubica e interconecta, mediante las herramientas de diseño, para conseguir el circuito deseado. Esta técnica puede alcanzar altas densidades de integración y excelentes prestaciones en velocidad y consumo de energía, y es considerada una solución intermedia entre el diseño *full-custom* y la realización a partir de matrices de puertas. La utilización de esta técnica repercute positivamente en el tiempo de desarrollo y disminuye el riesgo de fallos de fabricación, ya que el funcionamiento de los componentes elementales es comprobado previamente por el fabricante.

La tecnología basada en matrices de puertas parte de un diseño genérico, que previamente ha sido fabricado en silicio. Con esta alternativa es posible ahorrar tiempo de producción ya que las obleas permanecen en stock, a falta únicamente de especificar las capas de metalización que conectan los circuito. En esta tecnología, el diseñador elige el tipo de chip a utilizar, formado por matrices con celdas idénticas, y únicamente tiene que preocuparse de realizar las interconexiones entre las celdas, tanto a nivel interno como externo. Atendiendo a los recursos de interconexión y a la densidad de la matriz de puertas esta tecnología permite utilizar dos esquemas de matrices diferentes: con estructura acanalada (*gate array*) o con estructura sin acanalar (*sea of gate*). En el primer esquema (figura 2.1-c), la matriz cuenta con espacios o canales entre las celdas, lo que facilita la interconexión de los componentes. En el segundo (figura 2.1-d), el espacio entre las celdas no existe y las conexiones deben realizarse a través de las celdas no utilizadas.

La tecnología basada en matrices de puertas tiene un coste de producción inferior a los ASIC basados en celdas estándar y *full-custom*, debido a que sólo requieren de la fase de metalización. Por contra, presenta peores prestaciones en cuanto a velocidad y consumo de área de silicio, al ser diseños menos específicos y ser prácticamente imposible utilizar el 100 % de sus recursos. De hecho, con frecuencia, dada la dificultad del enrutado, suele ser necesario migrar a dispositivos con mayor densidad de celdas y, por tanto, de mayor coste económico.

Los diseños con matrices de puertas han evolucionando hacia el diseño de circuitos ASIC estructurados (figura 2.1-e), los cuales combinan las ventajas de los diseños basados en celdas estándar y los basados en matrices de puertas. Por un lado ofrecen la posibilidad de dedicar parte del área de silicio a núcleos IP (*Intellecual Property*) prediseñados, como, por ejemplo, procesadores, unidades DSP, memorias, etc., y por otro, incluyen una matriz de bloques lógicos que pueden ser configurados a partir de máscaras de metalización.

Los ASIC estructurados se diferencian de los circuitos de matrices de puertas en dos aspectos principales: que pueden utilizar bloques específicos IP y que pueden utilizar un conjunto de capas de metalización previamente diseñadas por el fabricante. Estas capas suelen incluir las conexiones de alimentación y los relojes del circuito, lo que supone una gran ventaja para el diseño al reducir el tiempo y el coste de fabricación. Esta tecnología dispone de herramientas de diseño y desarrollo más sencillas que las usadas por los circuitos de matrices de puertas,

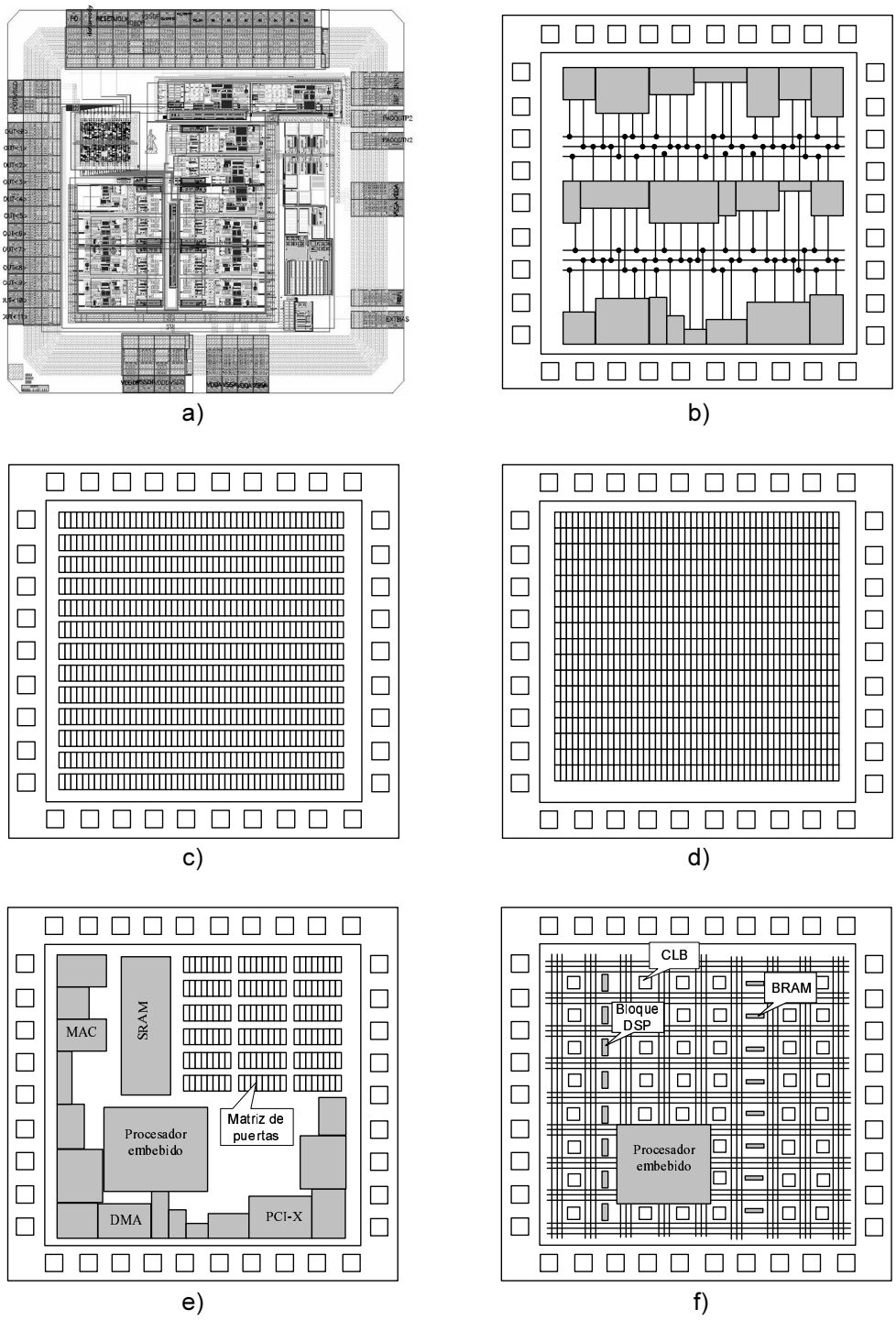


Figura 2.1: Tecnologias de fabricación de circuitos integrados.

mientras que sus prestaciones son casi tan buenas como las conseguidas por los circuitos basados en celdas estándar. Dado que para fabricar estos circuitos únicamente es necesario diseñar un pequeño número de capas metálicas, esta tecnología es considerada una opción intermedia entre la tecnología ASIC y las FPGA.

Para diseños puramente digitales, una alternativa que está tomando cada vez mayor auge es la utilización de dispositivos lógicos reconfigurables, como las FPGAs (*Fiel Programmable Gate Array*). Estos dispositivos (figura 2.1-f) se asemejan a los ASIC estructurados, ya que combinan área destinada a núcleos IP (memoria, procesadores, componentes DSP...) con área asignada a lógica configurable, sin embargo, tanto la tecnología de fabricación como el proceso de configuración son completamente diferentes. Mientras que los ASIC estructurados utilizan máscaras de metalización y solo pueden ser fabricados en foundry (plantas dedicadas a la fabricación de circuitos integrados), las FPGAs pueden ser configuradas de forma rápida y sencilla usando un simple PC.

Las FPGAs tienen la ventaja de poder ser reconfiguradas tantas veces como se desee y de ser reutilizadas en aplicaciones completamente distintas a las previstas originalmente. La reconfiguración de las FPGAs es un proceso sencillo, cuya duración depende de la densidad de puertas del dispositivo aunque no suele exceder de unos pocos minutos, lo que simplifica enormemente el ciclo del diseño. Por otro lado, las FPGAs tienen como desventaja que los circuitos que implementan presentan menor velocidad de funcionamiento, menor rendimiento de área y mayor consumo de energía que los implementados en los circuitos ASIC. La figura 2.2 (Xilinx, 2011) esquematiza las diferencias en los flujos de trabajo de diseños basados en FPGAs y los basados en circuitos ASIC.

Aunque inicialmente las FPGAs eran empleadas en diseños de baja velocidad, en la actualidad con ellas es posible alcanzar altas prestaciones de cómputo y frecuencia de reloj superiores a los 500 MHz. El aumento de la densidad de la lógica y las características de sus recursos internos (lógica de acarreo, PowerPC, bloques DSP, memorias internas, etc.) han permitido alcanzar altas velocidades de funcionamiento a precios cada vez más competitivos. La disminución en los precios de las FPGAs junto con su flexibilidad han permitido recortar terreno a los circuitos ASIC de forma rápida. El cuadro 2.1 muestra las principales ventajas de ambas tecnologías.

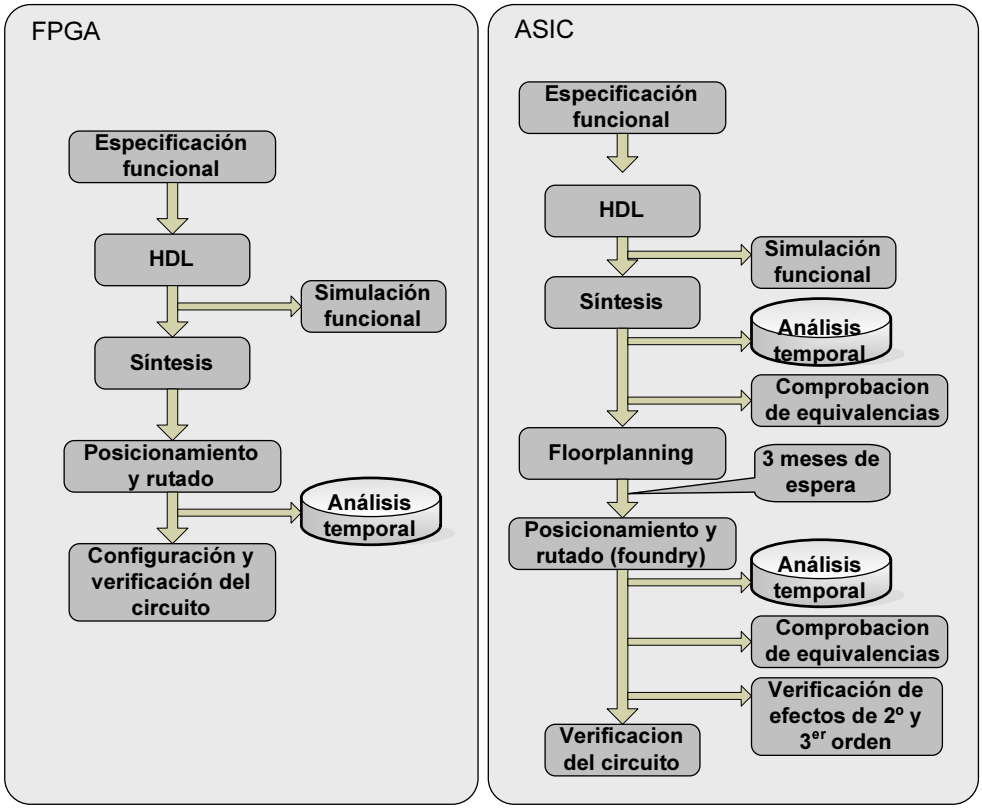


Figura 2.2: Flujos de trabajo en diseños basados en FPGAs y ASIC.

Cuadro 2.1: Ventajas de los diseños basados en FPGAs y ASIC

FPGA	ASIC
Se comercializan más rápidamente, no requieren máscaras ni medidas adicionales de fabricación	Diseño completamente a medida según especificaciones.
No conllevan gastos extraordinaros asociados normalmente al diseño ASIC (errores, dependencia con la tecnología...)	Costes unitarios reducidos para volúmenes de producción muy altos.
Ciclos de diseño más cortos gracias a que las herramientas automatizadas solucionan la mayor parte del emplazamiento, el rutado y las temporizaciones.	Dispositivos más eficientes en área/consumo.
Ciclos de producción más predecibles.	Mayores velocidad de reloj interno.
Capacidad para la reprogramación, volcado de nuevas configuraciones de forma remota.	

2.4. Implementación del circuito estándar de las redes CNN sobre ASIC

Cuando las redes CNN son implementadas en circuitos ASIC se parte de la base de que los circuitos serán más eficientes tanto en consumo de área de silicio como en consumo de energía. Si además los circuitos se implementan usando componentes electrónicos analógicos (Zarandy et al., 2002), la velocidad de procesamiento alcanzada será muy superior a la obtenida mediante circuitos electrónicos digitales. Esta ventaja es consecuencia de la respuesta prácticamente inmediata de los circuitos integradores analógicos, en comparación con los tiempos de integración de los métodos numéricos usados en los circuitos digitales.

En los siguientes apartados se describen los circuitos de referencia más utilizados en la realización de las redes CNN sobre circuitos ASIC analógicos: el circuito continuo y el circuito discreto.

2.4.1. Circuito estándar de las redes CNN continuas

En la figura 2.3 se muestra el esquema eléctrico de la celda CNN, tal y como fue propuesto originalmente por Chua y Yang (Chua L., 1988). El circuito de la celda $C(i,j)$ está compuesto por una fuente de tensión independiente U_{ij} (que representa la entrada U_{ij}), una fuente de corriente independiente I_{ij} (que representa el bias), tantas fuentes de corriente dependientes $I_{ukl}(i,j;k,l)$ e $I_{ykl}(i,j;k,l)$ como entradas externas u_{ukl} y de realimentación u_{ykl} tenga la celda, un condensador C y dos resistencia R_x y R_y. Las fuentes I_{ukl} e I_{ykl} dependen respectivamente de los valores de los operadores A y B y de las entradas u_{ykl} y u_{ukl} según las expresiones: $I_{ukl}(i,j;k,l) = B(i,j;k,l) \cdot u_{ukl}$ e $I_{ykl}(i,j;k,l) = A(i,j;k,l) \cdot u_{ykl}$, para cada $C(k,l) \in N_r(i,j)$. El único componente del circuito no lineal es la fuente de corriente controlada por tensión $I_y(i,j;k,l) = f(u_{xij})$ que, junto con la resistencia R_Y, caracterizan la salida de la celda según la expresión: $y_{ij} = (1/R_y)f(u_{xij})$, la cual representa la función de activación estándar de la celda 1.4 (capítulo 1). Aplicando las Leyes de Kirchhoff al circuito es posible obtener la ecuación diferencial que rige la dinámica de la celda, y que es representada en la ecuación 2.1:

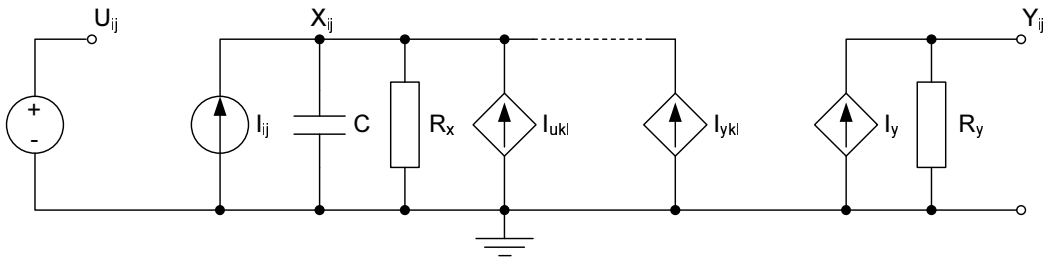


Figura 2.3: Diagrama eléctrico de la celda estándar de la red CNN continua.

$$C\frac{d}{dt}x_{ij}(t) = -\frac{1}{R_x}x_{ij}(t) + \sum_{k,l\in Nr(ij)} A_{kl}u_{ykl}(t) + \sum_{k,l\in Nr(ij)} B_{kl}u_{ukl}(t) + I_{ij} \qquad (2.1)$$

Donde x_{ij}, u_{ykl}, u_{ukl} son respectivamente la variable de estado y las entradas de tensión de la celda (i, j). La corriente de bias es representada por la constante I_{ij} y el producto R_xC define la constante de tiempo de integración del circuito que, en la mayoría de los casos y sin pérdida de generalidad, se hace coincidir con 1 para simplificar la ecuación.

2.4.2. Circuito estándar de las redes CNN discretas

La primera implementación *full-custom* de la red CNN fue realizada en 1992 utilizando tecnología CMOS de $4\mu m$ y el modelo discreto de la CNN (DT-CNN). En el artículo original de dicha propuesta (Harrer et al., 1992) se muestra el esquema eléctrico de la celda (figura 2.4) y algunos resultados de la implementación de una red CNN hexagonal de 16 celdas. El circuito discreto de la celda es obtenido a partir del circuito continuo, sustituyendo su variable de estado (x_{ij}) por una versión discreta $(v^x_{ij}(kT))$, obtenida a su vez al conmutar la carga y descarga de los condensadores C_1 y C_2 en intervalos de tiempo kT.

El circuito utiliza dos tipos de fuentes de corrientes dependientes: $I_{ykl}(kT)$, I_{ukl}, las cuales dependen, respectivamente, del producto de las entradas de la celda con los operadores $A(i, j; k, l)$ y $B(i, j; k, l)$. La resultante de la suma de estas corrientes más la suma de la corriente de bias I_{ij} ocasiona una caída de tensión en la resistencia R_x que determina el valor de la variable de estado discreta de la

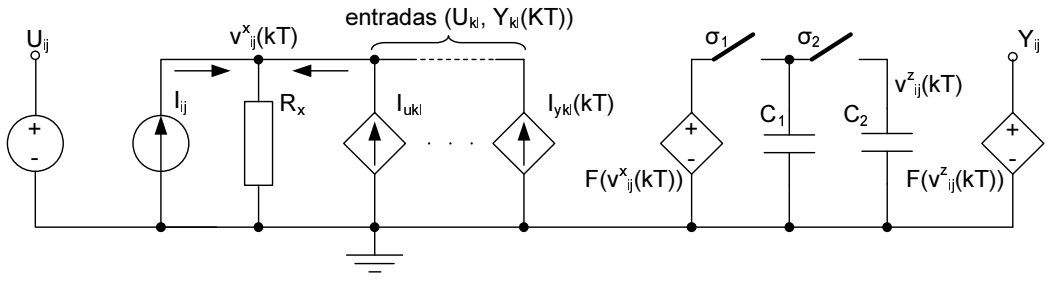


Figura 2.4: Diagrama eléctrico de la celda estándar de la red CNN discreta.

celda ($v_{ij}^x(kT)$), expresión 2.2.

$$v_{ij}^x(kT) = R_x \left(\sum_{k,l \in Nr(ij)} A_{kl} y_{kl}(kT) + \sum_{k,l \in Nr(ij)} B_{kl} u_{kl} + I_{ij} \right) \tag{2.2}$$

Las fuentes de tensión dependientes del circuito: $F\left(v_{ij}^x(kT)\right)$ y $F\left(v_{ij}^z(kT)\right)$, son utilizadas, respectivamente, para caracterizar la variable de estado de la celda y su salida. Dichas fuentes presentan un comportamiento no lineal, definido por la expresión 2.3.

$$F(v) = \begin{cases} +V_{sat} & \text{si} \quad v > 0 \\ -V_{sat} & \text{si} \quad v < 0 \end{cases} \tag{2.3}$$

La discretización de la variable de estado es obtenida a partir de un esquema de reloj de 2 fases no solapados (σ_1, σ_2). En la primera fase (σ_1), el condensador C_1 es cargado con el valor $+V_{sat}$ o $-V_{sat}$ en función del valor de la variable de estado en la iteración anterior ($v_{ij}^x((k-1)T)$). Durante la segunda fase (σ_2), los condensadores C_1 y C_2 son conectados entre sí para provocar entre ambos una transferencia de carga. Superado el régimen transitorio, el valor de tensión en el condensador C_2 ($v_{ij}^z(kT)$) puede ser calculado a partir de la exprexión 2.4, obtenida tras aplicar la ley de conservación de carga en los condensadores en ambas fases: $Q_{C_1C_2}^{\sigma_2} = Q_{C_1}^{\sigma_1} + Q_{C_2}^{\sigma_1}$.

$$v_{ij}^z(kT) = \frac{C_1 F(v_{ij}^x((k-1)T)) + C_2 v_{ij}^z((k-1)T)}{C_1 + C_2} \tag{2.4}$$

Escogiendo apropiadamente los valores de los condensadores $C1$ y $C2$, de manera que C_1 sea mayor que C_2, el signo de la tensión $v_{ij}^z(kT)$ vendrá determinado por la expresión $F(v_{ij}^x((k-1)T))$, ya que $\left|F(v_{ij}^x)\right| = V_{sat}$ y $\left|v_{ij}^z\right| \leq V_{sat}$ para todo

k. A partir de esta deducción, el valor de la salida Y_{ij} de la celda vendrá determinado por la expresión 2.5, que representa la función de activación hard limiter (figura1.6-c).

$$Y_{ij} = F(v_{ij}^z(kT)) = F(v_{ij}^x((k-1)T)) = \begin{cases} +V_{sat} & \text{si} \quad v_{ij}^x((k-1)T) > 0 \\ -V_{sat} & \text{si} \quad v_{ij}^x((k-1)T) < 0 \end{cases} \tag{2.5}$$

2.5. Arquitectura estándar de la red CNN-UM

La complejidad del diseño microelectrónico ocasiona que la realización de las redes CNN como circuito integrado sea una tarea difícil y con grandes limitaciones tecnológicas. Las primeras realizaciones ASIC de las redes CNN implementaban estructuras simples, de una sola capa, que siempre ejecutaban la misma tarea. En algunos casos, aunque los circuitos podían ser reprogramados, el tiempo requerido para ello, superior en varios órdenes de magnitud al del propio procesamiento, limitaba esta posibilidad e impedía sacar beneficio de las ventajas de los circuitos analógicos.

Este inconveniente, junto con la necesidad de algunas aplicaciones de ejecutar algoritmos compuestos por varias plantillas, fue la principal razón que dio lugar al desarrollo de las redes CNN Universal Machine (CNN-UM) (Roska y Chua 1993). Las redes CNN-UM se basan en una arquitectura que permite combinar operaciones analógicas con operaciones booleanas y llevar a cabo diferentes tareas de procesamiento en un mismo chip. La arquitectura fue diseñada para llevar a cabo una tarea de procesamiento y, a continuación, reintroducir nuevamente el resultado en el circuito para volverlo a procesar. En cada iteración, las celdas pueden ser reprogramadas con plantillas diferentes para realizar tareas de procesamiento diversas. La concatenación secuencial de varias tareas de procesamiento permite que la red CNN-UM pueda desarrollar algoritmos complejos y abordar gran cantidad de aplicaciones. Con esta arquitectura, la reprogramación de las celdas es realizada en tiempos comparables al del procesamiento por lo que el ciclo de operación no se ve penalizado excesivamente.

Las redes CNN-UM pueden ser desarrolladas a partir del circuito continuo de la red CNN (sección 2.4.1) o a partir del circuito discreto (sección 2.4.2). En ambos casos, y dada la ventaja que supone para la implementación que la

variable de estado esté acotada a límites seguros, como sucede en el modelo de rango completo (sección 1.2.3), la resistencia R_x de los circuitos (figuras 2.3 y 2.4) ha sido sustituida por una resistencia de tipo no lineal. Otra característica de las CNN-UM es que la funcionalidad de las celdas ha sido extendida, con respecto a la celda tradicional, con nuevos componentes analógico y digital (memorias y unidades de procesamiento). Además, para garantizar la reprogramación de las celdas, la arquitectura ha sido equipada con una unidad de programación global y con lógica de control adicional.

En la figura 2.5 se observa que la arquitectura genérica de la CNN-UM está compuesta por dos bloques: La matriz de celdas extendidas y la unidad analógica/digital de programación global (GAPU, *Global Analogic Programming Unit*). En la misma figura se muestra el esquema interno de la celda extendida.

La celda extendida está compuesta por un núcleo de procesamiento estándar, un conjunto de circuitos locales y dos clases de memorias diferentes: la memoria analógica local (LAM, *Local Analog Memory*) y la memoria lógica local (LLM, *Local Logical Memory*), usadas, respectivamente, para almacenar información analógica y digital. La memoria LAM se implementa mediante condensadores y se distribuye sobre cuatro zonas del núcleo de procesamiento para almacenar distintos tipos de datos: el valor de entrada de la celda V_{uij} (en LAM1), el estado inicial $V_{xij}(0)$ (en LAM2), el bias (en LAM3) y la secuencia de salidas de la celda V^k_{yij} (en LAM4). Cuando el algoritmo de procesamiento ejecuta secuencialmente diferentes etapas de procesamiento, los resultados de la salida, almacenados en la memoria LAM4, son realimentados a la memoria LAM1 para reintroducirlos nuevamente como entradas. Por otra parte, la memoria LLM es implementada usando registro de desplazamiento y es utilizada para almacenar la salida de la celda una vez ha sido digitalizada (Y^k_{Lij}).

Los circuitos de procesamiento incluidos en celda extendida son: la unidad de salida analógica local (LAOU, *Local Analog Output Unit*) y la unidad lógica local (LLU, *Local Logic Unit*), ambos disponen de múltiples entradas y una única salida. La unidad analógica LAOU recibe como entradas los datos almacenados en la memoria LAM4 y su salida es una combinación lineal de dichas entradas. Esta unidad realiza operaciones analógicas simples, del tipo sumas y restas, y los resultados son convertidos a binario y almacenados en la memoria LLM. Por otro lado, la unidad LLU dispone de una funcionalidad similar a la de la unidad

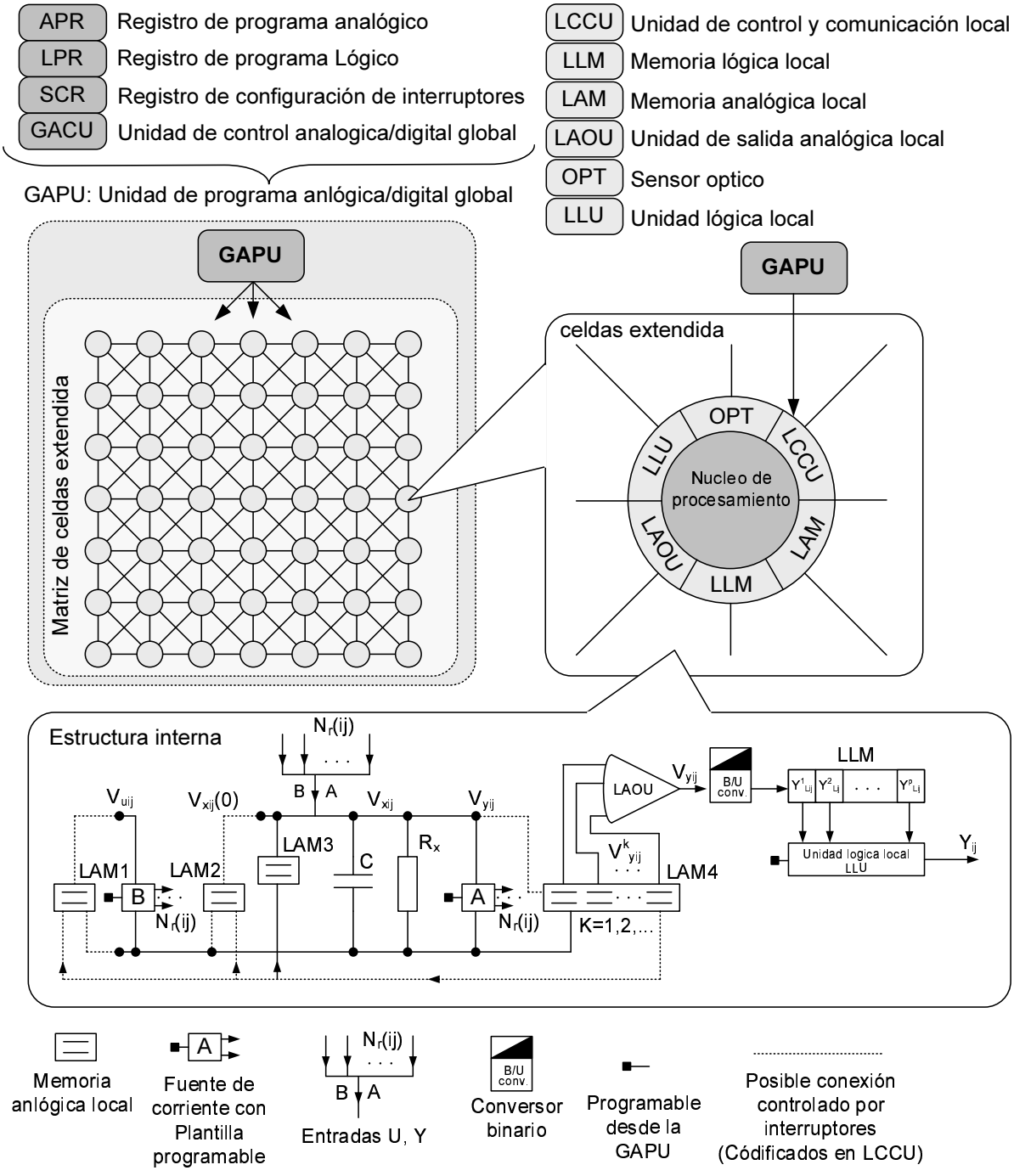


Figura 2.5: Arquitectura estándar de la CNN-UM y circuito interno de la celda extendida.

LAOU, aunque en este caso de carácter digital. La unidad LLU puede realizar operaciones booleanas sencillas del tipo: AND, OR, XOR, etc. Hay que indicar que las operaciones realizadas por estos dos elementos de procesamiento pueden llevarse a cabo de forma convencional por el núcleo de la celda, sin embargo, el uso especifico de estos elementos incrementará la eficiencia del sistema.

Otro elemento incluido en la celda, es la unidad de control y comunicación local (LCCU, *Local Communication and Contro Unit*). Este elemento encargado de establecer la comunicación entre la unidad de programación analógica/digital global de la red (GAPU) y el núcleo de procesamiento de la celda. La LCCU recibe instrucciones globales desde la GAPU y permite que la celda pueda ser reprogramada. La información recibida por la LCCU contiene: el valor de la plantilla analógica (operadores A, B e I), el código de la función lógica de la unidad LLU,y la configuración interna del núcleo de procesamiento (interconexionado, operaciones de la LAOU, etc.). Dado que cada celda dispone de su propia unidad LCCU, la GAPU puede realizar la reprogramación todas las celdas de la red de forma simultánea y a alta velocidad, lo que supone una de las principales ventajas de las redes CNN-UM.

Finalmente, otro elemento que suele añadirse a las celdas de las redes CNN-UM, aunque no todas cuentan con él, es el sensor óptico (OPT, *Optical Sensor*), utilizado para adquirir directamente la información de las imágenes que se desean procesar.

El otro bloque importante que constituye la arquitectura de la CNN-UM, y cuyo uso es compartido por todas las celdas de la matriz, es la GAPU. Esta unidad está formada por cuatro elementos funcionales, la unidad de control GACU (*Global Analogic Control Unit*) y tres registros: APR (*Analog Program Register*), LPR (*Logic Program Register*) y SCR (*Switch Configuration Register*). Estos tres registros son utilizados por la GAPU para almacenar temporalmente la información que se envía a las celdas y que es recibida por las unidades LCCU de cada una de ellas. El registro APR contiene el código de la plantilla que usarán las celdas en la siguiente iteración del procesamiento. El registro LPR almacena el código de la función lógica de la unidad LLU, mientras que el registro SCR almacena la configuración interna del núcleo de procesamiento. Finalmente, la unidad de control GACU tiene como misión almacenar, decodificar y cargar en los tres registros anteriores, la secuencia de tareas o instrucciones que ejecutará la CNN-UM.

La combinación de elementos analógicos y digitales, junto a la capacidad de reprogramación, hace que las redes CNN-UM sean vistas como computadores mixtos, denominados *analogic computer*. En este sentido, cada etapa de procesamiento es interpretada como una instrucción (analógica o digital) que es almacenada junto a otras para constituir programas que son ejecutados para realizar determinadas tarea de procesamiento. Se han desarrollado multitud de plantillas y combinaciones de instrucciones, que se han recopilado en diferentes bibliotecas (Roska et al., 2000), que permiten realizar multitud de operaciones orientadas al procesamiento de imágenes. Como ya se ha indicado, además, existen diferentes herramientas de apoyo, como compiladores, sistemas operativos y simuladores, que dan soporte a las redes CNN-UM y que permiten gestionar sus recursos hardware.

2.5.1. Realizaciones hardware de las redes CNN-UM

Desde que en 1993 se implementara por primer vez el núcleo de las redes CNN-UM (Harrer et al., 1993) gran cantidad de chips han sido diseñados con el fin de introducir nuevas características o mejoras a este tipo de redes. En los siguientes apartados se analizan las principales implementaciones operativas realizadas sobre circuitos ASIC.

Familia de chips ACE

La familia de chips ACE comenzó a desarrollarse en 1994 en el Centro Nacional de microelectrónica (Sevilla) por un equipo dirigido por Rodríguez-Vazquez. Estos dispositivos de señal mixta cuentan con las características funcionales y elementos estructurales de la arquitectura estándar de las redes CNN-UM, revisada en la sección anterior (sección 2.5).

Los chips de la familia ACE disponen de una matriz de celdas idénticas que incorporan localmente, un sensor óptico, diferentes memorias tipo LAM y LLM, unidades de procesamiento LLU y LAOU y un núcleo de procesamiento analógico de radio de vecindad 3×3. Estos elementos pueden ser reprogramados desde una unidad de control global, que proporciona gran versatilidad a la arquitectura y que permite que pueda abordar multitud de aplicaciones. Todas las celdas

son gestionadas por dicha unidad de control y son configuradas simultáneamente con la misma instrucción. La unidad de control se encarga de secuenciar y temporizar las instrucciones almacenadas en memoria, junto con los parámetros de configuración internos de las celdas.

El primer chip operativo de esta familia, denominado ACE400 (Domínguez et al., 1997), es una versión revisada y mejorada del prototipo preliminar de 1994 (Domínguez et al., 1994). El ACE400 fue fabricado por primera vez en 1996, usando tecnología CMOS de $0.8\mu m$, e incorporaba una matriz de 440 celdas (20×22) y diferentes circuitos auxiliares: interfaces E/S, una unidad de control, memorias de coeficientes, convertidores que traducen la información de programación en señales analógicas, etc. Originalmente, el chip tenía una capacidad de programación bastante limitada y solamente podía trabajar con imágenes binarias (en blanco y negro). En su desarrollo se puso especial interés en la interfaz óptica de la entrada, la cual fue diseñada con amplificadores de alta ganancia para incrementar la velocidad de captura de las imágenes.

El siguiente chip de la familia ACE, denominado ACE4K (Liñán et al., 2002), comenzó a desarrollarse en el año 2000 e incorporando una matriz de 64×64 celdas y mayor capacidad de programación. A pesar de contar con más celdas que su predecesor (ACE400), ser más complejo y poder procesar imágenes en escala de grises (codificados en 4 bits), su consumo de energía se redujo manteniendo la misma velocidad de procesamiento. Estas características fueron conseguidas gracias a una mejora en la arquitectura y en la circuitería interna de la celda, así como a una reducción en la escala de integración del chip, pasando de un proceso de fabricación de $0.8\mu m$ a uno de $0.5\mu m$.

La última versión de esta familia ha sido el chip ACE16k (Liñán et al., 2003), que terminó de desarrollarse a finales de 2002. Esta nueva versión, fabricada con tecnología CMOS de $0.35\mu m$, dispone de una matriz de 128×128 celdas y puede operar con imágenes digitalizadas de 8 bits, alcanzando, en conjunto, con el 85 % de sus componentes trabajando de forma analógica, un pico de cómputo de hasta 330 GOPS. Las mejoras de este chip con respecto a la versión ACE4K, incluyen la incorporación de buses digitales, mejoras en el control de acceso a los datos de las interfaces E/S y óptica y modificaciones en la organización interna de las unidades de procesamiento. Las memorias LLMs son sustituidas por memorias tipo LAMs (más rápidas), se mejora la gestión del consumo y se incrementa la capacidad

de la memoria para almacenar programas. Más detalles sobre las características internas de la arquitectura y sus mejoras pueden ser revisadas en (Liñán 2002).

Otro chip fabricado por el mismo equipo de investigadores, aunque desarrollado para una aplicación específica, es el CACE1K (Carmona et al., 2003). Este chip, diseñado en 2003, fue concebido para emular el procesamiento espacio-temporal realizado por la retina del ojo. Su arquitectura, similar a la del chip ACE4K, cuenta con una estructura interna más compleja formada por dos capas de celdas interconectadas entre sí.

En la figura 2.6 se muestran las conexiones sinápticas de la red CNN tanto a nivel interno (en una sola capa) como entre las dos capas. Cada celda recibe la contribución de otras celdas situadas en su misma capa (en un entorno de radio de vecindad 1) y de una única celda situada en la otra capa. Las constantes de integración de cada capa son diferentes, siendo la de la primera variable y programable mediante 4 bits de configuración (t_1) y la de la segunda prefijada a un determinado valor constante (t_2). Al igual que ocurre en el chip ACE4K cada celda incluye una unidad LLU y memorias del tipo LAMs y LLMs. Estos componentes, junto con los parámetros de configuración internos del núcleo de las celdas, pueden ser reprogramados a través una unidad de programación global. El chip CACE1K fue diseñado con tecnología CMOS de $0.5\mu m$ e incorpora en cada capa una matriz de 32×32 celdas.

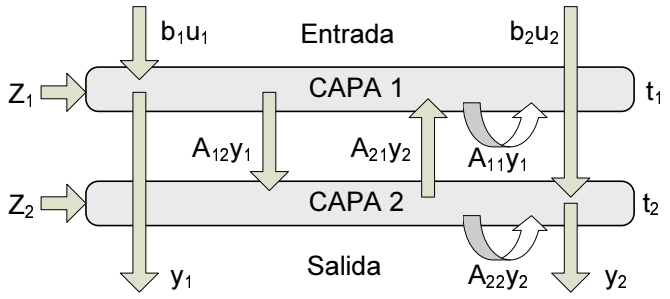


Figura 2.6: Esquema de interconexión de las capas CNN del chip CACE1K.

Familia de chips Helsinki

La familia Helsinki, desarrollada en Finlandia en el laboratorio de diseño de circuitos electrónicos de la universidad de Helsinki, presenta una arquitectura

interna distinta a la utilizada por las redes CNN-UM estándar. Aunque el núcleo de procesamiento es similar al utilizado por la celda clásica, los elementos locales (memorias, unidades de procesamiento, unidades de comunicación) se distribuyen de manera diferente y presentan distinta funcionalidad.

El esquema de la figura 2.7, muestra los bloques funcionales de la arquitectura interna de las celdas de la familia Helsinki. Dichas celdas han sido diseñadas para interconectarse en un radio de vecindad uno. De forma general, la arquitectura dispone de cuatro memorias locales, una unidad lógica programable y varios interruptores que controlan el modo de funcionamiento de la celda. Una característica que diferencia a esta arquitectura es su salida, la cual coincide con una función de tipo sigmoidal de gran ganancia (figura 1.6-c), que se implementa mediante un comparador de corriente. La arquitectura Helsinki fue diseñada principalmente para desarrollar aplicaciones de segmentación de imágenes en blanco y negro.

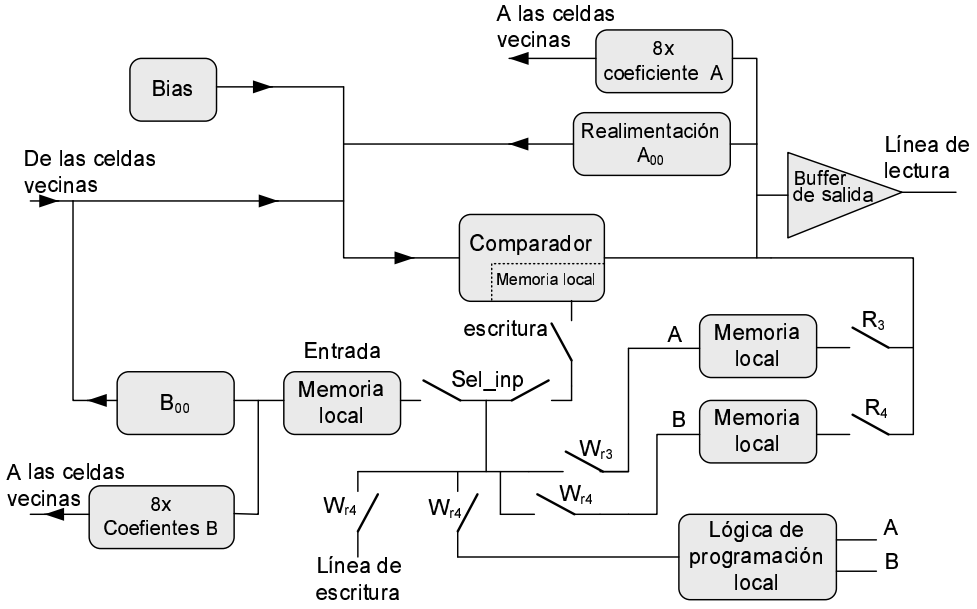


Figura 2.7: Diagrama de bloques de la arquitectura Helsinki.

El primer chip de la familia Helsinki (Paasio et al., 1997) fue desarrollado en 1997 con tecnología CMOS de $0.8\mu m$ e incluía una matriz de 16×16 celdas. En la siguiente versión del chip, del año 1998 (Paasio et al., 1998), la matriz fue extendida a 48×48 celdas y se fabricó con tecnología de $0.5\mu m$. Un año más tarde, la arquitectura Helsinki fue utilizada para desarrollar la estructura

con mayor densidad de celdas realizada hasta el momento (Paasio et al., 1999), compuesta por 176×144 celdas coincidiendo con el tamaño de las imágenes del formato de vídeo estándar QCIF. Está densidad de integración se obtuvo a partir de un tecnología de fabricación de $0.25\mu m$.

2.5.2. Limitaciones en la implementación analógica de las redes CNN

Los circuitos ASIC analógicos pueden ser considerados como la solución más eficiente para implementar las redes CNN, principalmente debido a su menor consumo de área de silicio y a su mayor velocidad de procesamiento. No obstante, esta solución presenta importantes inconvenientes que dificultan la realización de estos sistemas, los más importantes son: su complejidad y tiempo de desarrollo, su alto coste de fabricación y su escasa flexibilidad para el diseño.

El coste económico que supone la realización de los circuitos ASIC hace que, en ocasiones, su fabricación sea difícil de justificar. En muchos casos, esta solución solamente es viable cuando, por razones de requerimientos de velocidad y densidad de integración, no existe otra alternativa para la implementación. Por otro lado, la complejidad que supone la realización de los circuitos ASIC, junto con su falta de flexibilidad, hace que realmente sea complicado implementar redes CNN complejas, como las compuestas por redes multicapa o las que utilizan plantillas no-lineales, dependientes o variables en el tiempo.

Los circuitos analógicos utilizados por las redes CNN presentan también inconvenientes relacionados con el proceso de fabricación: parásitos, tolerancias, defectos, sensibilidad al ruido y a cambios de temperatura, etc. Estos problemas ocasionan que los cálculos realizados por dichos circuitos presenten errores de precisión y que, por consiguiente, no puedan ser utilizados en muchas aplicaciones. Debido a la importancia de este asunto, algunas líneas de investigación se han centrado concretamente en dar solución a este tipo de problemas.

En este sentido, se ha tratado de mejorar la precisión de las redes CNN analógicos incrementado la robustez de las plantillas, de forma que estas puedan absorber las dispersiones que se producen el proceso de fabricación (George y Moschytz, 1999). Por otro lado, se han desarrollado plantillas óptimas diseñadas específi-

camente para aplicaciones y chips determinados (Xavier-de Souza et al., 2003; Xavier-de Souza et al., 2004). En otros casos se ha tratado de mejorar el comportamiento de los circuitos analógicos utilizando semiconductores tipo BJT (*Bipolar Junction Transistor*) (Wen y Chung 2000) o basados en diodos túnel, con menor sensibles al proceso de fabricación que los transistores CMOS.

Otra alternativa en la que se ha trabajado, es la realización óptica de la red CNN, tanto continua (Sullivan et al., 1996) como discreta (Buczynski et al., 1998). Con esta solución se pretende mejorar la conexión de las entradas y salidas de las celdas para hacerlas menos sensibles a las distorsiones y diafonías ocasionadas por las interferencias electromagnéticas (Tokes et al., 2003).

Finalmente, existen iniciativas para tratar de desarrollar nuevas arquitecturas que reduzcan los requerimientos de área e interconexionado de las redes. Este es el caso de las redes en topología en estrella, las cuales utilizan un solo multiplicador multiplexado en el tiempo para realizar el procesamiento (Sargeni et al., 2006).

2.6. Realización de las redes CNN mediante dispositivos electrónicos digitales

Como respuesta a los inconvenientes de la implementación analógica de las redes CNN, algunos investigadores han evaluado diferentes soluciones hardware basadas en circuitos electrónicos digitales. Estas implementaciones presentan grandes ventajas aunque también importantes inconvenientes, como una mayor ocupación de área de silicio, un mayor consumo de energía y una menor velocidad de funcionamiento. La realización de la redes CNN mediante circuitos digitales tiene como principales ventajas el uso de un sistema de codificación mucho más robusto y mayor nivel de precisión en los resultados, lo que permite utilizar las redes CNN en aplicaciones de resolución de ecuaciones diferenciales y de procesamiento de imágenes en escala de grises. Otras importantes ventajas de los circuitos digitales son la flexibilidad que presentan para el diseño y la utilización de herramientas de desarrollo más sencillas y fáciles de automatizar. En particular, las herramientas de síntesis electrónica basada en lenguajes de descripción a nivel de sistema (ESL, *Electronic System Level*) permiten realizar la síntesis de algoritmos de alta complejidad de forma automática.

2.7. Arquitectura Castle

La arquitectura Castle (Zarandy et al., 1998) fue desarrollada en 1998 para resolver, mediante circuitos ASIC digitales, la ecuación de estado del modelo discreto de las redes CNN. Dicho modelo hace uso de la condiciones de contorno de flujo cero (Nuemann), de forma que el estado de las celdas frontera debe ser doblado en el contorno de la matriz de celdas.

Para simplificar las operaciones y reducir el área del circuito, la variable de estado del modelo ha sido acotada al intervalo de entrada ($[-1, +1]$), de forma similar a como se hace en el modelo de señal de rango acotado FSR-CNN (sección 1.2.3). Otra simplificación de la arquitectura Castle es la separación de la ecuación de estado del modelo en dos términos semejantes: el término 2.6 y el término 2.7, ambos formados por un sumatorio de productos y una suma.

$$g_{ij} = \sum_{k,l \in Nr(ij)} B_{kl} U_{kl}(n) + I_{ij} \tag{2.6}$$

$$X_{ij}(n+1) = \sum_{k,l \in Nr(ij)} A_{kl} X_{kl}(n) + g_{ij} \tag{2.7}$$

En esta aproximación, la entrada U y el operador B son considerados constantes, por tanto, el término g_{ij} únicamente tiene que ser calculado una vez. Esta aproximación tiene como objetivo simplificar las operaciones del modelo: en la primera iteración se calcula el término constante (ecuación 2.6) y en el resto, solamente, el término dependiente de la variable de estado (ecuación 2.7).

La arquitectura Castle (figura 2.8) puede hacer uso de varios Elementos de Procesamiento (EPs), organizados en columnas, para incrementar la velocidad de procesamiento del sistema. Cada columna de EPs se encarga de procesar una banda vertical de píxeles, cuya anchura varía en función del tamaño de la imagen. La unidad de control global de la arquitectura, implementada mediante un circuito de reloj de dos fases no solapadas, se encarga de temporizar y sincronizar el procesamiento. La imagen es introducida secuencialmente en la primera fila de EPs que, tras calcular el término 2.6, envía el resultado a la siguiente fila de EPs, donde se inician los cálculos del término 2.7. Para implementar las operaciones de las ecuaciones 2.6 y 2.7, los EPs cuentan con una unidad de procesamiento aritmética, una unidad de control y dos memorias: la matriz de registros y la memoria de plantillas.

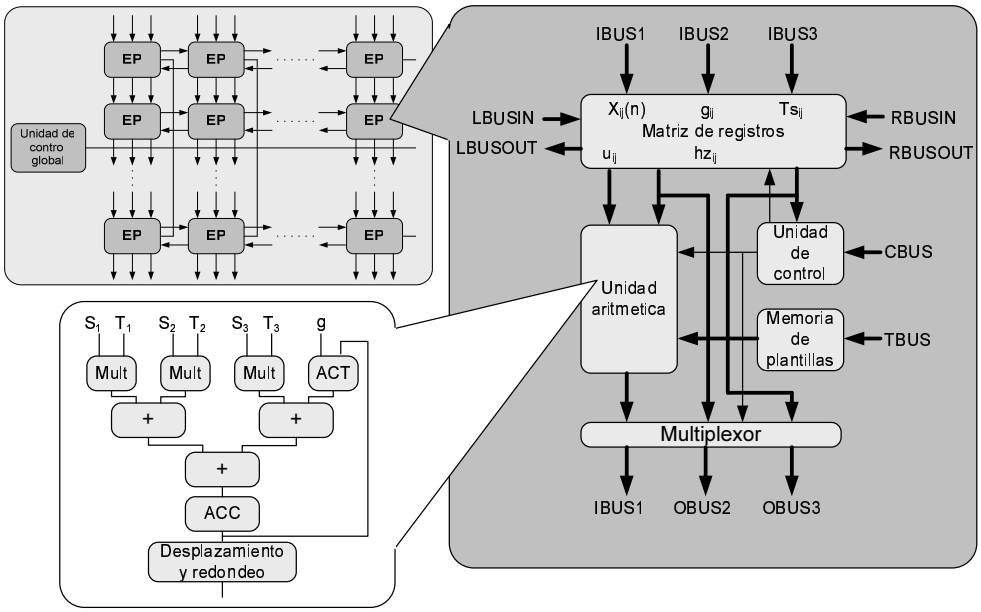


Figura 2.8: Diagrama de bloques de la arquitectura Castle.

La matriz de registros almacena a través de los buses IBUS1, IBUS2 e IBUS3, respectivamente, la variable de estado de la celda X_{kl} (la entrada U_{kl} en el caso de la ecuación 2.6), la constante g_{kl} (el bias I_{ij} en el caso de la ecuación 2.6) y el código de selección de la plantilla T_{Sij}. La memoria de plantillas hace uso del bus TBUS para almacenar 16 plantillas diferentes, cada una de 9 coeficientes. La unidad de control, a través del bus CBUS, determina el modo de funcionamiento del EP, que puede ser configurado en función de tres niveles de precisión: 1 bit, 6 bits y 12 bits. La unidad aritmética, esquematizada en la parte izquierda de la figura 2.8, utiliza internamente 3 multiplicadores, 3 sumadores y 2 registros acumuladores para ejecutar secuencialmente las operaciones en 3 ciclos de reloj, consumiendo un total 24 ns según los cálculos del autor (Zarandy et al., 1998). Finalmente, en el esquema se muestran los cuatro buses (LBUSIN, RBUSIN, RBUSOUT y RBUSOUT) utilizados por el EP para establecer la comunicación con los situados a su izquierda y a su derecha.

Para comprender el funcionamiento de la arquitectura Castle es necesario entender la estructura interna de su matriz de registros, la cual, como se muestra en la figura 2.9-a, está compuesta por cuatro filas de 40 registros cada una y dos columnas laterales con 3 registros.

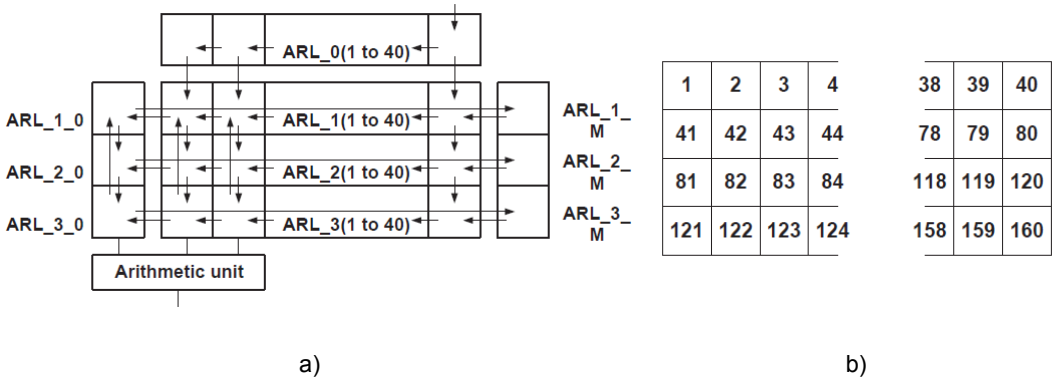


Figura 2.9: Estructura de la matriz de registros de la arquitectura Castle. a) Matriz con 40 registros por fila. b) Píxeles de la imagen de test.

La primera fila de la matriz de registros, denominada $ARL-0$, se utiliza para almacenar temporalmente los 40 últimos datos que se reciben por la entrada del EP. Las otras tres filas: $ARL-1,2,3$, almacenan los 40×3 datos anteriores. Las columnas $ARL-10$ y $ARL-1M$ se usan para duplicar los valores de los EPs vecinos y mantener la continuidad en el procesamiento, o en el caso de que los EPs sean frontera, para copiar los valores de los registros adyacentes y establecer las condiciones de contorno Neumann. Para una red CNN de radio de vecindad 1, el tamaño de la matriz de registros (Tm) vendrá dado por la expresión 2.8 (Nagy 2007):

$$Tm = w(4 \cdot sw + 3(cw + tsw)) \tag{2.8}$$

Donde, la variable w representa el número de elementos de cada fila, y las variables sw, cw y tsw respectivamente, la anchura en bits de la variable de estado X_{ij}, la constante g_{ij} y el código de selección de la plantilla (Ts_{ij}). A partir de la expresión 2.8, y considerando que tanto la variable de estado como la constante g_{ij} son de 12 bits y el código de selección de plantillas de 4 bits, se deduce que la matriz de registros hará uso de un total de 3840 registros de 1 bit.

2.7.1. Secuencias de carga y procesamiento de la arquitectura Castle

La figura 2.10 muestra la secuencia llevada a cabo por la arquitectura Castle para procesar una imagen de tamaño 40×4 píxeles (figura 2.9-b). Dicha secuencia

comienza con la carga de datos en la matriz de registros. La carga se realiza a través del registro situado más a la derecha de la fila $ARL - 0$, introduciendo la información en serie a la vez que el contenido de dicha fila se desplaza una posición hacia la izquierda. Cuando la fila $ARL - 0$ termina de completarse, toda la información se desplaza una posición hacia abajo para ocupar la siguiente fila de registros $ARL - 1$ (paso 41 de la figura 2.10). En el caso de que los datos almacenados en $ARL - 0$ representen los 40 primeros píxeles de la imagen, su contenido es desplazado dos posiciones hacia abajo (hasta la fila $ARL - 2$) para garantizar las condiciones de contorno Neumann. La secuencia se volverá a repetir con los siguientes 40 píxeles de la imagen, aunque a partir de ahora, cada vez que la fila $ARL - 0$ se complete, la información se desplazará solamente una posición hacia abajo. Tras el segundo desplazamiento, la secuencia de carga inicial de la matriz de registros finaliza (paso 83), quedando la primera línea de píxeles de la imagen almacenada en las filas $ARL - 3$ y $ARL - 2$ y la segunda en la fila $ARL - 1$.

Tras la carga inicial, la secuencia continúa con la fase de procesamiento de los datos almacenados en las tres filas inferiores de la matriz de registros. Esta fase consiste en realizar, 40 veces por fila (120 veces), las operaciones indicadas en la ecuación 2.7. Previamente, si el EP no tiene vecinos laterales, las columnas de registros $ARL - 10$ y $ARL - 1M$ son cargadas con los valores de los extremos de las filas de registros $ARL - 1, 2, 3$ (paso 84), con el fin de poder establecer las condiciones de contorno Neumann.

Como se muestra en la figura 2.9-a, la unidad aritmética se encuentra conectada a los tres últimos de registros de la izquierda de la fila inferior de la matriz de registros. Para que la unidad aritmética pueda cargar los datos almacenados en las dos filas superiores, se realizan 3 ciclos consecutivos en los que se rota el contenido de las tres filas de registros, que finalmente dejan la información en su posición original (pasos 85-87). En la siguiente iteración (paso 88), la información de la matriz de registros es desplazada una posición hacia la izquierda, a la vez que se introduce un nuevo dato en la fila $ARL - 0$.

Una vez procesado el contenido de la matriz de registros (pasos 89-244), la secuencia continúa con la introducción de dos pasos extra que dejan nuevamente la información en su posición original (pasos 245-246). En el siguiente paso (paso 247), los nuevos píxeles que han sido introducidos en la fila $ARL - 0$ son

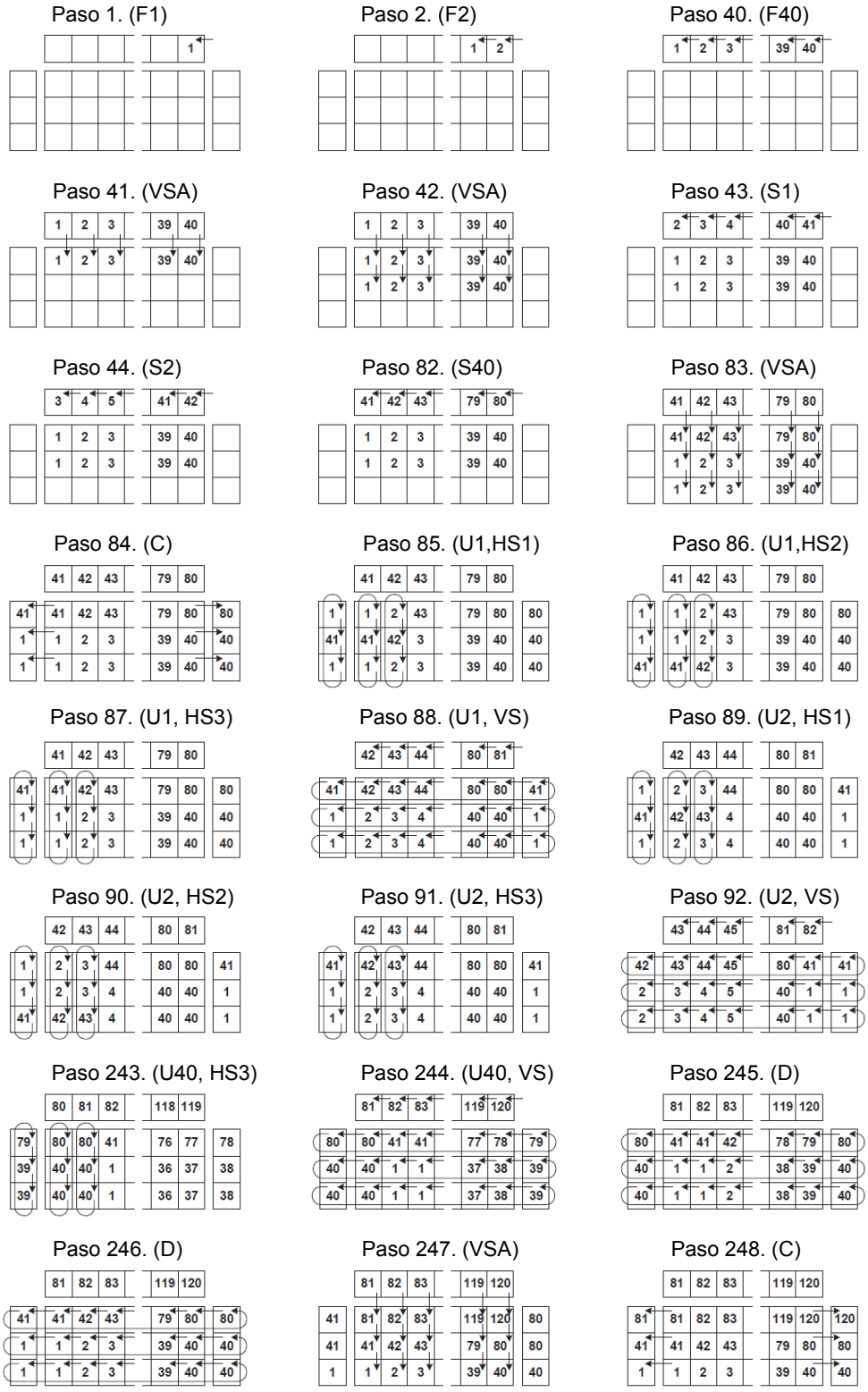


Figura 2.10: Flujo de carga de datos y procesamiento de la arquitectura Castle.

desplazados una posición hacia abajo para poder repetir nuevamente el proceso. La figura 2.11 muestra el diagrama de estados de la arquitectura Castle para la secuencia de carga y procesamiento descritos.

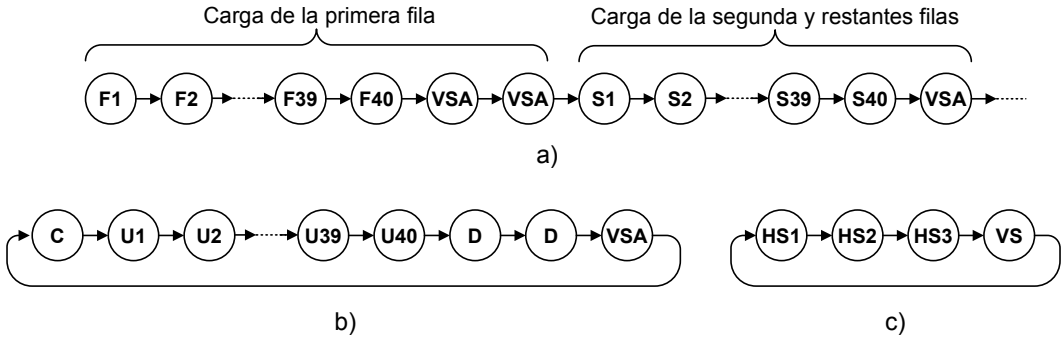


Figura 2.11: Diagrama de estados de la arquitectura Castle. a) Diagrama correspondiente a la carga de las dos primeras filas de datos. b) Diagrama de estados del procesamiento de una fila. c) Diagrama de procesamiento de un dato.

Teniendo en cuenta las figuras 2.10 y 2.11, y considerando que el número de filas de una imagen es $f = 4$ y el de elementos por fila $d = 40$, se puede determinar que la carga inicial de la primera fila de la matriz de registros consume $d+2 = 42$ pasos (1-42) y que la carga del resto de filas consume $d + 1 = 41$ pasos (43-83). De igual forma, y dado que para procesar un dato son requeridos 4 pasos (por ejemplo, los 85-88 para el dato U1), se deduce que el número total de pasos para procesar una fila completa es igual a $4 \cdot d + 4 = 164$ pasos (84-247).

A partir de estos resultados se puede concluir que el número de ciclos de reloj necesarios para carga una imagen puede determinarse a partir de la expresión 2.9 ($NCc = 165$), y que el número total de ciclos requeridos para procesar una imagen puede calcularse a partir de la expresión 2.10 ($NCp = 656$).

$$NCc = (d + 2) + (f - 1) \cdot (d + 1) \tag{2.9}$$

$$NCp = (4 \cdot d + 4) \cdot f \tag{2.10}$$

2.8. Realización de redes CNN sobre dispositivos lógicos reconfigurables

En los últimos años, la tendencia seguida en la realización hardware de las redes CNN ha sido la utilización de dispositivos lógicos reconfigurable (FPGAs). Las FPGAs, a pesar de trabajar con frecuencias de reloj un orden de magnitud inferior al de los microprocesadores convencionales, suponen una alternativa interesante al desarrollo de sistemas en tiempo real debido a su gran capacidad para el procesamiento paralelo. Las ventajas que ofrecen las FPGAs para el diseño (flexibilidad, reducido coste, ahorro de tiempo, etc.), junto con las características de sus herramientas de desarrollo, hacen que esta tecnología sea una alternativa excelente para emular y acelerar aplicaciones basadas en redes CNN.

2.9. Arquitectura Falcon

La arquitectura Falcon comenzó a desarrollarse en el año 2002 (Nagy y Szolgay 2003; Nagy 2007) con el objetivo de incrementar la flexibilidad de la arquitectura Castle y mantener sus prestaciones de cómputo. La arquitectura Falcon fue diseñada originalmente para ser implementada sobre FPGAs.

La arquitectura resuelve el modelo de Euler de la red CNN usando la aproximación FSR (Full Singal Range) en la que se considera que tanto las variables de estado como las de salida de las celdas se encuentran acotadas en el intervalo $[-1, +1]$, ecuaciones 2.11 y 2.12.

$$X_{ij}(n+1) = X_{ij}(n) + \sum_{k,l \in Nr(ij)} A_{kl} X_{kl}(n) + g_{ij} \tag{2.11}$$

$$g_{ij} = \sum_{k,l \in Nr(ij)} B_{kl} U_{kl}(n) + I_{ij} \tag{2.12}$$

En esta aproximación, los operadores A y B de la plantilla incluyen el valor de la constante de muestreo h del sistema y, a diferencia de lo sucede en la arquitectura Castle, la ecuación 2.11 mantiene el término $X_{ij}(n)$, lo que permite minimizar el número de bits utilizados para almacenar el elemento central del operador A.

Como sucede en la arquitectura Castle, la arquitectura Falcon considera que la entrada U y el operador B del modelo son constantes y que, por tanto, el término g_{ij} (ecuación 2.12) solamente debe ser calculado una vez, al principio del procesamiento.

La figura 2.12 representa el esquema general de conexión de la arquitectura Falcon y los detalles principales de su estructura interna. Como se observa, la unidad aritmética ha sido modificada, con respecto a la arquitectura Castle, incorporando registros adicionales entre los sumadores y segmentando los multiplicadores para incrementar la velocidad de procesamiento.

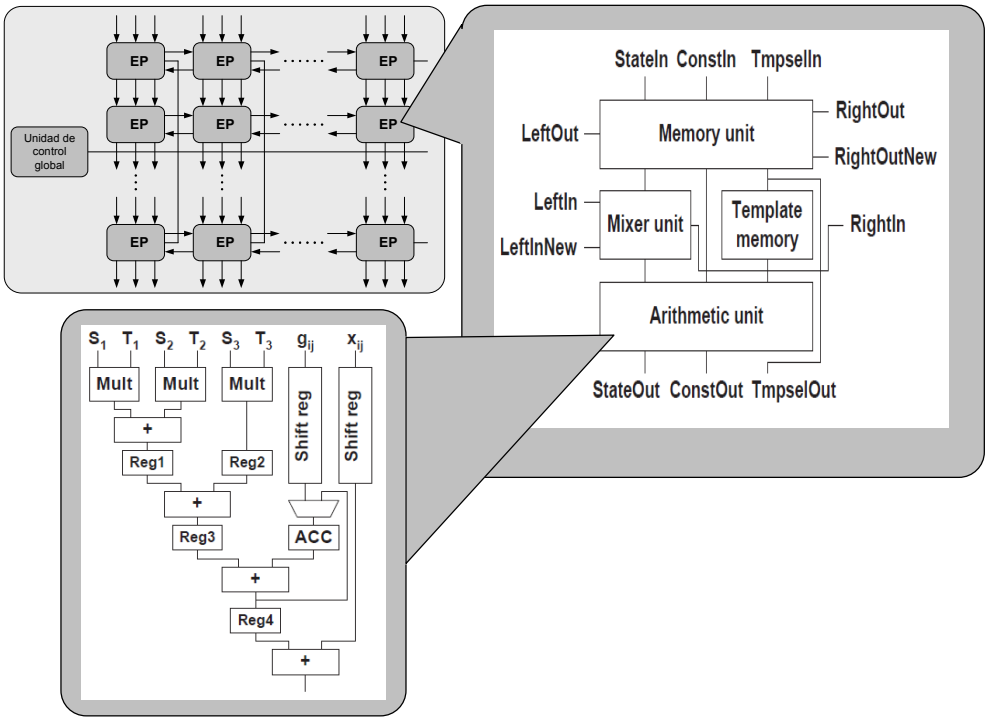


Figura 2.12: Diagrama de bloques de la arquitectura Falcon.

Cada columna de Elementos de Procesamiento (EPs) recibe una banda vertical de la imagen que es almacenada internamente de forma similar a como se hace en la arquitectura Castle. Aunque en este caso, la memoria utilizada es de menor tamaño y su estructura interna diferente. Cada uno de los EP de la arquitectura dispone internamente de 2 tipos de memoria: una denominada memoria principal, que almacena tres filas de la banda de la imagen, y la otra denominada unidad mezcladora, que almacena los datos de la ventana actual de procesamiento.

Las condiciones de contorno Neumann de la primera y última fila de la imagen se implementan en la memoria principal cargando, simultáneamente, los mismos datos de entrada en la primera y segunda fila de la memoria.

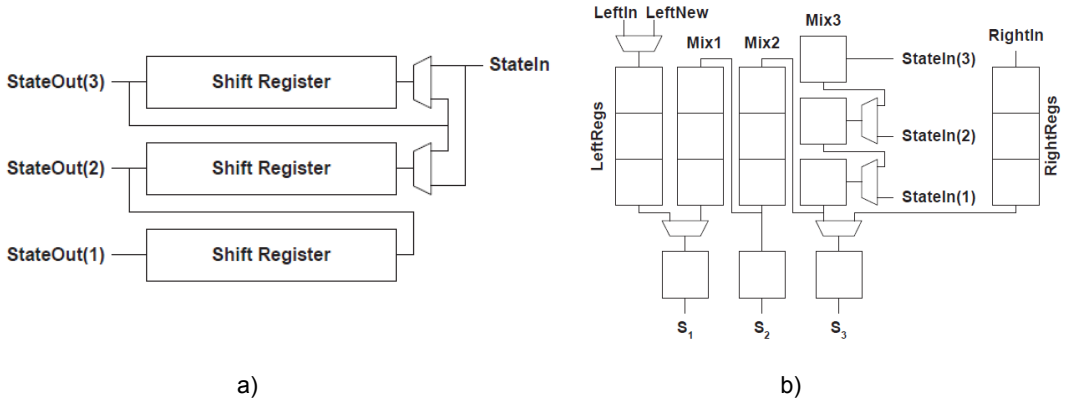


Figura 2.13: Memoria de la arquitectura Falcon para un radio de vecindad $r = 1$. a) Memoria principal. b) Unidad mezcladora.

En la memoria principal de la arquitectura Falcon se han eliminado, con respecto a la arquitectura Castle, una fila de registros completa y las dos columnas de registros laterales. Para una red CNN de radio de vecindad uno, el tamaño de la memoria principal (Tm) puede ser calculado a partir de la expresión 2.13 (Nagy 2007), donde w es el número de elementos de cada fila y sw, cw y tsw, respectivamente, la anchura en bits de la variable de estado, la constante g_{ij} y el código de selección de las plantillas.

$$Tm = w(3 \cdot sw + 2 \cdot (cw + tsw)) \tag{2.13}$$

Como se observa en la figura 2.13-a, las filas de la memoria principal están compuesta por un conjunto de w registros de desplazamiento serie. La entrada de la memoria principal (StateIn) está conecta a la entrada de las dos primeras filas de registros para poder establecer los valores de contorno Neumann en las fronteras superior e inferior de la imagen. Por otra parte, la salida de cada fila (StateOut 1,2,3) está conectada a la entrada de la fila inmedianteamente inferior y a las correspondientes entradas de la unidad mezcladora (StateIn 1,2,3).

Para un radio de vecindad uno, la unidad mezcladora (figura 2.13-b) está compuesta por cinco columnas de registros de sw bits. La columna Mix3 contiene

tres registros de carga paralela y salida serie. Las columnas Mix1 y Mix2 están formadas por tres registros de desplazamiento serie que, junto con la columna anterior, se utilizan para almacenar la información de la ventana de vecindad que va a ser procesada. Las otras dos columnas de registros, denominadas LeftRegs y RighRegs, se utilizan para almacenar los datos compartidos con los EPs vecinos, de la izquierda y de la derecha, o en el caso de que no hayan, para completar las fronteras laterales de las condiciones de contorno Neumman. La características de la unidad mezcladora, junto con la utilización de una unidad aritmética con 3 multiplicadores, permiten procesar la ventana de vecindad 3×3 en 3 ciclos de reloj, lo que da un factor de utilización de los multiplicadores del 100 %.

2.9.1. Arquitectura Falcon multi-capa con GAPU

La arquitectura Falcon ha sido extendida al caso multi-capa para poder emular redes CNN con varias capas paralelas, donde las variables de estado son sumadas entre sí para determinar la respuesta de la red. Las expresiones 2.14 y 2.15 representan el modelo de Euler de esta aproximación, siendo p la variable que determina el número de capas.

$$X_{m,ij}(n+1) = X_{m,ij}(n) + \sum_{q=1}^{p} \sum_{k,l \in Nr(ij)} A_{mq,kl} \cdot X_{q,kl}(n) + g_{m,ij} \tag{2.14}$$

$$g_{m,ij} = \sum_{q=1}^{p} \sum_{k,l \in Nr(ij)} B_{mq,kl} \cdot U_{q,kl}(n) + h \cdot I_{q,ij} \tag{2.15}$$

Como se observa en la figura 2.14-a, la extensión de la arquitectura Falcon implica la utilización de p memorias y p unidades de mezcla idénticas a las definidas en la sección anterior. Cada capa utiliza una unidad aritmética (figura 2.14-b) que se conecta a las salidas de todas las unidades de mezcla, lo que supone un consumo p veces superior de multiplicadores con respecto a la arquitectura de una capa.

La arquitectura Falcon multi-capa ha sido extendida con una Unidad de Programación Analógica/Digital Global (GAPU) (Voroshazi et al., 2008) que se encarga de secuenciar el juego de plantillas y de ejecutar un número arbitrario de iteraciones de Euler. Dicha unidad incluye además diferentes registros de estatus y de comandos.

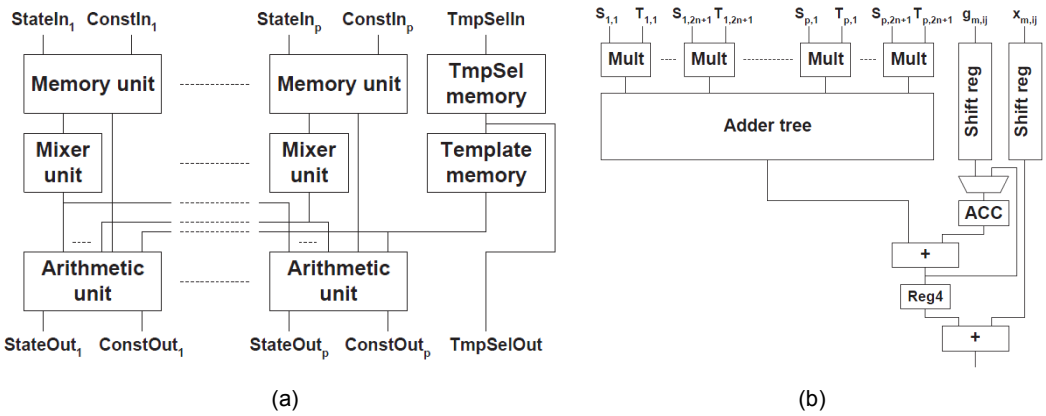


Figura 2.14: Memoria de la arquitectura Falcon para un radio de vecindad uno. a) Memoria principal. b) Unidad mezcladora.

2.10. Características de las arquitecturas Castle y Falcon

Las arquitecturas Castle y Falcon han sido diseñadas para ser implementadas mediante dispositivos electrónicos digitales y resolver la aproximación de Euler del modelo de la red FSR-CNN, considerando condiciones de contorno Neumann y que tanto el operador B de la plantilla como la entrada U del modelo son constantes. Las aproximaciones utilizadas para simplificar el modelo implican que las aplicaciones desarrolladas con estas arquitecturas deberán incluir al menos 2 EPs por columna para completar una iteración de Euler. El primer EP calcula el término constante g_{ij} mientras que el segundo actualiza la variable de estado X_{ij} del modelo.

Las dos arquitecturas pueden hacer uso de varias columnas de EPs para acelerar el procesamiento, cada una de ellas ejecuta simultáneamente una banda vertical de la imagen. Los procesadores de la fila superior de las columnas reciben las bandas de la imagen (las variables de estado X_{ij}), el término constante g_{ij} y el término de selección de plantillas Ts_{ij}, y devuelven un resultado que es enviado al siguiente EP de la columna.

El esquema de procesamiento con varias flujos paralelos permite incrementar el rendimiento de las arquitecturas, aunque también es difícil de aprovechar debido al gran ancho de banda requerido y a la complejidad que supone el diseño de la

interfaz de entrada y salida. Este problema se incrementa conforme aumenta el número de columnas de EPs y el tamaño de las imágenes a procesar, lo que a su vez reduce la capacidad de adaptación de la arquitectura y fuerza al desarrollo de redes CNN de topología simple. La escalabilidad de estas arquitecturas es reducida cuando se añaden nuevos flujos de procesamiento debido a que esto supone el rediseño completo de la interfaz de memoria y de su unidad de gestión.

Capítulo 3

Planteamiento del sistema, arquitectura de cómputo y discretización del modelo

En este capítulo se plantean las principales especificaciones y restricciones del sistema utilizado para verificar, analizar y dar soporte a la arquitectura hardware que emula a las redes CNN. Se expondrá el marco de trabajo empleado para simplificar dicho sistema y se definirá el concepto de arquitectura de cómputo, especificando los niveles de abstracción que han sido utilizados para organizar y estructurar su diseño. Por otro lado, se propondrá el conjunto de requisitos deseables de la arquitectura de cómputo y se establecerán las principales reglas de diseño utilizadas para su desarrollo.

Debido a los altos requisitos temporales y de velocidad exigidos por las aplicaciones de tiempo real, durante el diseño de la arquitectura ha sido necesario considerar y tomar como referencia algunas características de las principales arquitecturas de cómputo paralelas. Por esta razón, y con el fin de facilitar la identificación del paralelismo aplicado a nuestra arquitectura, se ha visto conveniente realizar una breve introducción a los conceptos básicos sobre paralelismo y sobre las arquitecturas de cómputo paralelas.

En la parte final de este capítulo se muestra la metodología utilizada para

desarrollar la arquitectura y sistematizar el proceso de diseño e implementación del hardware. El proceso de diseño parte en este capítuo con el planteamiento de los diferentes métodos de discretización que se han utilizado para transformar el modelo continuo de la red CNN en diferentes aproximaciones discretas. Dichas aproximaciones son a continuación simuladas con el propósito de comprobar su funcionalidad y determinar sus requerimientos computacionales. Finalmente, la aproximación con mejores prestaciones es elegida para desarrollar la arquitectura de cómputo y llevar a cabo la implementación.

3.1. Características del sistema

El principal objetivo de este trabajo es diseñar e implementar sobre hardware reconfigurable una arquitectura de cómputo que permita emular en tiempo real el comportamiento de una red CNN. Hay que tener en cuenta, sin embargo, que dicha arquitectura formará parte de un sistema mayor que se utilizará para verificar y dar soporte a su funcionalidad. Las características y funcionalidad de este sistema vendrán determinadas por el modelo de red CNN y por el tipo de aplicaciones a las que se destine. Ambos aspectos, modelo y campo de aplicación, deberán ser claramente especificados con el fin de concretar los detalles de la realización del sistema y determinar, entre otras cosas, qué componentes hardware serán necesarios para la implementación, cuál será su disposición y conexionado y qué prestaciones deberán tener.

Una de las características más deseables del sistema es que sea de propósito general, es decir, que cuente con las características básicas necesarias para llevar a cabo un amplio conjunto de tareas dentro de su campo de aplicación. Lo que se pretende es que el sistema sea útil y que responda a ciertas necesidades que justifiquen su implementación y puesta en funcionamiento.

A continuación, se expone el marco de trabajo utilizado para simplificar el sistema y permitir que su realización hardware pueda ser abordada de forma cómoda y sencilla. El objetivo que se persigue es reducir el abanico de posibles soluciones que se plantean durante el diseño a partir de la especificación de su funcionalidad básica y de su campo de aplicación.

Elección del modelo de la red CNN

La primera consideración para simplificar el sistema ha sido especificar el modelo de red CNN que será utilizo para define la arquitectura de cómputo del sistema. Para ello, se ha tratado de seleccionar una aproximación discreta lo más sencilla posible y que permita maximizar la eficiencia computacional. Dicha elección está motivada por el interés de implementar la arquitectura sobre hardware reconfigurables minimizando el consumo de recursos de cómputo. Para obtener la aproximación discreta se ha utilizado como referencia el modelo estándar de la red CNN (Chua L., 1988) por su carácter generalista.

Hay que indicar que la discretización del modelo continuo de la red CNN puede limitar la funcionalidad del sistema e impedir que éste pueda ser utilizado en determinadas aplicaciones, como, por ejemplo, las que dependen de los efectos térmicos o de las interferencias electromagnéticas que se producen en el interior de los circuitos electrónicos analógicos (Brown y L., 1992). Además, hay que tener en cuenta que el modelo de referencia utilizado no incorpora la funcionalidad de otros modelos, como por ejemplo el retraso de las redes D-CNN, lo que impedirá que el sistema pueda ser utilizado en aplicaciones de carácter temporal, dedicadas al análisis de movimiento y de velocidad.

Estas limitaciones, sin embargo, no restarán generalidad al desarrollo y podrán ser tenidas en cuenta en futuros trabajos para mejorar las prestaciones y características del sistema. En general, en la mayoría de las aplicaciones a las que destina el sistema, la utilización de un modelo discreto aproximado de la red CNN no supondrá mayor inconveniente que el asociado con la cuantización y discretización de las variables.

Elección del nicho de aplicación

Otro aspecto tenido en cuenta para simplificar el sistema ha sido establecer su nicho de aplicación. Los procesos de diseño y realización hardware de un sistema de cómputo son tareas complejas que requieren el uso de ciertas restricciones para su simplificación. En nuestro caso, dichas restricciones estarán determinadas por el campo de aplicación y servirán para desarrollar únicamente aquellas funciones del sistema que sean imprescindibles para el funcionamiento de las aplicaciones.

En nuestro caso, el sistema se ha planteado para ser utilizado en aplicaciones de procesamiento de imagen y vídeo en tiempo real, dada la adecuación de las redes CNN a este tipo de aplicación. En particular, y sin que esto suponga una pérdida de generalidad, el sistema ha sido diseñado para trabajar con redes CNN de radio de vecindad 1 e imágenes de hasta 1024×1024 píxeles codificadas con 256 niveles de grises. Esta especificación tiene como principal objetivo simplificar y optimizar la implementación hardware del sistema. Por otro lado, hay que indicar que con estas restricciones será posible abordar aplicaciones de mayor tamaño, mayor resolución y precisión que las realizadas mediante sistemas CNN analógicos basados en circuitos ASIC tradicionales.

Para verificar y controlar el funcionamiento de las aplicaciones, ha sido necesario equipar al sistema con componentes auxiliares, como un sensor de imágenes CMOS, un convertidor D/A, un monitor VGA, etc. Algunos de esto componentes, como ocurre en el caso del *frame grabber*, han tenido que ser diseñados e implementados a la medida de las necesidades de nuestro sistema.

3.2. Arquitectura de cómputo del sistema

En las siguientes secciones se exponen las principales características de la entidad del sistema encargada de llevara a cabo el procesamiento. Para ello, en primer lugar, definiremos lo que entenderemos por *arquitectura de cómputo* del sistema.

La Arquitectura de Computadores es el área encargada de abordar el diseño conceptual y la estructura operacional fundamental de los sistemas de cómputo. Las principales definiciones sobre el concepto de arquitectura de un computador tratan, fundamentalmente, la descripción de los recursos hardware/software que son necesarios para definir una máquina de cómputo de propósito general, del tipo: Von-Newman, harvard, RISC, etc. El marco de estas definiciones abarca, por consiguiente, tanto los niveles tecnológicos de la realización del hardware como los niveles del sistema operativo y del software, incluyendo los compiladores y los lenguajes de programación.

El concepto de arquitectura de cómputo que aplicaremos nosotros tratará exclusivamente la descripción de la estructura y la organización interna de los

circuitos electrónicos encargados de realizar el procesamiento del sistema, es decir, no tendrá en cuenta los aspectos relacionados con el software. Dicha arquitectura estará especializada en la resolución de un problema específico y será analizada y diseñada tanto desde el punto de vista de su *microarquitectura* e *implementación lógica* como desde el punto de vista de los recursos tecnológicos necesarios para su realización. Entendiendo por *microarquitectura* el conjunto de recursos y métodos utilizados para satisfacer las especificaciones del hardware, y por *implementación lógica* la definición y realización de los circuitos lógicos necesarios para satisfacer las especificaciones de dicha microarquitectura.

Bajo estas consideraciones definiremos el concepto de arquitectura de cómputo de nuestro sistema como *"la entidad electrónica encargada de almacenar, procesar y transferir la información que recibe el sistema. Dicha entidad será concretada mediante una serie de componentes y conexiones hardware, que serán definidos a partir de una estructura y una organización jerárquica que dependerá tanto del modelo computacional de la red CNN como de un conjunto de requisitos funcionales, de restricción y de calidad."*

3.2.1. Niveles de abstracción de la arquitectura de cómputo

La arquitectura de cómputo del sistema heredará, a través de la especificación de la microarquitectura de sus unidades de procesamiento, la estructura y la topología implícita de la red CNN. Dado que interesa que la arquitectura sea genérica y lo más versátil posible, con el fin de poder ser utilizada en el mayor número de aplicaciones, por usuarios sin conocimientos en electrónica, el diseño de la arquitectura ha sido divido en dos partes diferenciables: la arquitectura externa y la arquitectura interna.

La arquitectura externa será definida por el usuario del sistema en función de las características de la red CNN y de los requisitos y necesidades de cada aplicación. Por otra parte, la arquitectura interna dará soporte a la parte externa mediante la especificación de la microarquitectura de sus unidades básicas del procesamiento, las cuales representan el medio físico a partir del cual la funcionalidad del sistema puede ser trasladada al hardware de forma transparente para el usuario.

Junto con la división anterior, la arquitectura del sistema ha sido organizada en base a una distribución jerárquica de niveles de abstracción, que permite abordar el estudio y el desarrollo de la arquitectura de manera estructurada, haciendo mucho más tratable los problemas de diseño y de implementación.

Teniendo como referencia la distribución de niveles de abstracción de diferentes sistemas de cómputo de propósito general (Ortega et al., 2005; Bell y Newell 1971; Tanenbaum 2000; Almeida et al., 2008), a continuación se especifica la distribución de niveles que se ha utilizado para diseñar nuestra arquitectura.

Cada nivel de abstracción puede ser interpretado desde dos puntos de vista diferentes: uno en función de su estructura y organización, y otro en función de su descripción funcional o comportamental. En la descripción estructural, los niveles son especificados en términos de componentes y conexiones, mientras que en la descripción funcional, la descripción es realizada en base a un concepto de caja negra, la cual queda definida a partir de su interfaz de E/S y de la información que el diseñador necesita para utilizarla en niveles superiores.

- **Nivel de sistema**. En este nivel, la red neuronal es tratada como un componente más conectado al sistema. La conexión se realiza mediante una sencilla interfaz de E/S que facilita la comunicación de los componentes y su sincronización. Dicha interfaz ha sido diseñada para ser flexible y versátil y poder aprovechar, de forma eficiente, tanto la capacidad de cómputo de la red CNN como la funcionalidad de los diferentes componentes del sistema. Los elementos estructurales de este nivel incluyen los dispositivos FPGAs utilizados para implementar la red CNN y los elementos auxiliares que dan soporte al desarrollo de las aplicaciones: cámaras de vídeo, *frame graber*, monitor de vídeo VGA, etc.

- **Nivel de red neuronal**. Los componentes estructurales de este nivel constituyen la entidad de procesamiento del sistema, la cual podrá estar constituida por una o varias redes CNN. Dichos componentes son definidos en los niveles inferiores y su interconexión permitirá establecer la topología explícita de la red CNN. La descripción funcional de este nivel viene determinada por la interfaz de E/S de la entidad de procesamiento y por la tarea de procesamiento que realiza. En este nivel, se considera que la entidad de procesamiento del sistema podrá estar distribuida en una o varias FPGAs.

- **Nivel de red de procesadores (Capa/Celda/Etapa)**. La descripción estructural de este nivel incluye los componentes y las conexiones necesarias para desarrollar la estructura neuronal de la red CNN (Celdas, Etapas, ruta de datos, componentes de unión, etc.). La arquitectura abstracta vendrá representada por la interfaz de E/S de los siguientes componentes: Capa, Celda y Etapa, y por el tipo de procesamiento que realizan: difusión, extracción de bordes, umbralización, perfilado, etc.

- **Nivel de operador**. La estructura y organización de este nivel viene determinada por la conexión de los componentes que constituyen la unidad básica de procesamiento, que se ha denominado Etapa. En este nivel, los componentes están representados por: unidades MAC (*Multiplier and Accumulator*), sumadores, función de activación, conexiones, etc. La descripción funcional de este nivel se define mediante la interfaz de E/S del componente Etapa y su funcionalidad, la cual se expresa en términos de su plantilla (A, B e I). La Etapa es el elemento de más bajo nivel que el usuario podrá utilizar para desarrollar la arquitectura externa de la red CNN.

- **Nivel de transferencia de registro**. La descripción estructural de este nivel viene determinada por componentes funcionales del tipo: contadores, multiplexores, sumadores, decodificadores, memorias, etc., previamente definidos en los niveles de abstracción inferiores. En este nivel, la información pasa de unos registros a otro mediante buses de conexión y circuitos combinacionales, donde es transformada y encaminada hacia otros componentes. La descripción funcional de este nivel es especificada mediante la interfaz de E/S de los operadores de cálculo de la Etapa.

- **Nivel de lógica digital**. La descripción estructural está formada por componentes cuyo comportamiento viene definido por las leyes del álgebra de Boole, se incluyen circuitos combinacionales, secuenciales y sus conexiones (puertas, triestados, biestables, buffer, osciladores, buses, memorias, señales, etc). La funcionalidad de los circuitos combinacionales vendrá especificada en términos de tablas de verdad o funciones lógicas, y la de los circuitos secuenciales mediante diagramas o tablas de estado. La arquitectura abstracta de este nivel vendrá definida por la interfaz de E/S de los circuitos que agrupan a estos componentes, su funcionalidad aritmética o lógica y

sus prestaciones (latencia, throughput, recursos hardware consumidos, velocidad de procesamiento, etc.).

- **Nivel tecnológico**. La descripción estructural de este nivel hace referencia a los componentes tecnológicos utilizados para llevar a cabo la implementación física de los circuitos electrónicos del sistema, siendo en nuestro caso los recursos específicos de las FPGAs: Slices (que incluyen LUTs, lógica de acarreo rápido, biestables...), DSP48 (unidades de procesamiento digital), BRAM (bloques de memoria), DCM (módulos generadores de reloj), matrices y lineas de conexión, buffers, triestados, puertos I/O, etc. Estos componentes son utilizados para construir y dar funcionalidad a todos los componentes lógicos del nivel superior.

La distribución de niveles propuesta (representada en la figura 3.1) tiene como objetivos facilitar el diseño, la implementación, la verificación y la utilización de la arquitectura de cómputo del sistema. En cada nivel, estos aspectos son tratados independientemente con el fin de aprovechar, con la mayor eficiencia posible, tanto los recursos hardware disponibles (capas, celdas, operadores de cómputo, decodificadores, puertas lógicas etc.) como el potencial de las herramientas de diseño y síntesis hardware utilizadas.

En los tres niveles inferiores, los componentes de la arquitectura son diseñados desde un punto de vista cercano al hardware, primando el uso eficiencia de los recursos de cómputo y la velocidad de procesamiento de los circuitos.

En el nivel de operador, los componentes de los niveles inferiores son organizados y conectados, de la forma más eficiente posible, siguiendo el flujo de datos del algoritmo que define la unidad básica de procesamiento de la red CNN, es decir, la Etapa.

En los tres niveles superiores, el usuario tendrá una visión funcional y operativa de la arquitectura que, independiente del hardware, permitirá aprovechar los recursos de bajo nivel para implementar aplicaciones de forma sencilla y eficiente.

Esta filosofía de diseño, estructurada en distintos niveles de abstracción jerárquicos, permitirá tanto al diseñador como al usuario organizar la arquitectura del sistema de forma adecuada y optimizar sus capacidades y recursos para conseguir las máximas prestaciones.

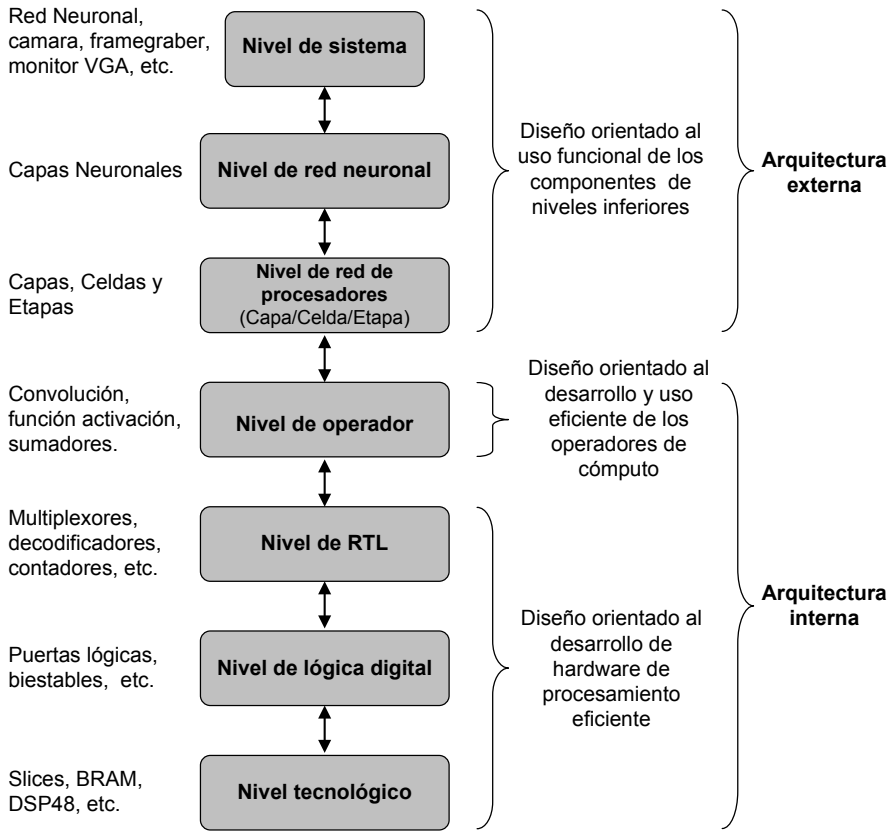


Figura 3.1: Niveles de abstracción del sistema. En la parte izquierda se indican los componentes que son utilizados en cada nivel. En la parte derecha se indica la orientación que se le ha dado al diseño.

3.3. Requisitos de diseño de la arquitectura de cómputo

A partir de un conjunto de requisitos de diseño, bien definido y preciso, es posible especificar las características deseables o exigibles a una arquitectura de cómputo y a su interacción con el entorno. Aunque es posible imaginarse multitud de requisitos, encontrar un conjunto adecuado, manejable y que englobe todas las características deseables es una tarea compleja y crítica que requiere gran experiencia y altos conocimientos sobre problema a resolver. A continuación, se indican los requisitos de diseño que se han tenido en cuenta para desarrollar nuestra arquitectura.

Requisitos estructurales

Estos requisitos especifican las necesidades de la arquitectura en cuanto a las características básicas de su microarquitectura. Las principales prestaciones de la arquitectura dependerán fundamentalmente de estos requisitos.

- **Recursos de cálculo**. Este requisito especifica el tipo y la cantidad de elementos de cálculo que son utilizados para llevar a cabo las operaciones que realiza la arquitectura. Es necesario contar al menos con un número mínimo de estos recursos para satisfacer, en el intervalo de tiempo exigido, las necesidades de cómputo del sistema. Dado que la arquitectura trabaja en tiempo real, es preciso que estos recursos estén sincronizados y que las operaciones se ejecuten en el tiempo establecido para no violar los requerimientos temporales.

- **Memoria**. Principalmente determina la capacidad de almacenamiento de la arquitectura. La memoria es utilizada para almacenar la información de entrada y salida requerida por las unidades de procesamiento. Además de la capacidad de almacenamiento, en el diseño se ha considerado: el tiempo de acceso de la memoria, el tamaño de palabra, la jerarquía de memoria, la organización (compartida, distribuida), el tipo de acceso (asíncrono o síncrono), el número de puertos (simple, múltiple acceso), etc.

Requisitos organizativos

Estos requisitos definen la estrategia que se seguirá a la hora de utilizar y combinar los diferentes recursos estructurales de la arquitectura. Entre otros aspectos permiten especificar la distribución de los componentes, su interacción, su versatilidad y modo de operación.

- **Modularidad**. Un aspecto interesante de la arquitectura es que pueda ser dividida en sub-bloques y que estos puedan ser desarrollados y ejecutados de manera independiente. La modularidad es esencial para simplificar el proceso de diseño de la arquitectura y las aplicaciones y llevar a cabo el mantenimiento y la depuración de errores.

- **Flexibilidad**. Este requisito mide la facilidad que supone modificar la arquitectura para afrontar cambios en las condiciones iniciales del diseño, por ejemplo: cambios en la precisión de las variables, modificar el algoritmo de procesamiento, cambios en la unidad de control, etc. Este requisito es importante en las fases preliminares del diseño, antes de que la arquitectura haya sido definida completamente.

- **Sencillez**. Esta característica mide el nivel de organización interna, física y lógica de la arquitectura para hacer frente a las tareas que tiene encomendadas. En general, cuanto más sencilla sea una arquitectura, más económica, rápida y eficiente será.

- **Escalabilidad**. Esta característica determina la capacidad de la arquitectura para ser ampliada con nuevos bloques funcionales, módulos o tareas sin que esto afecte gravemente a su eficiencia y coste (computacional, de diseño, de mantenimiento, etc.). Cumplir este requisito es importante para garantizar la adaptación de la arquitectura a las circunstancias cambiantes de diferentes aplicaciones.

- **Ampliabilidad**. Este requisito determina si físicamente es posible añadir más módulos a la arquitectura. Que la arquitectura sea ampliable no significa que sea escalable, es decir, la ampliabilidad no implica que la eficiencia tenga que mantenerse ni que su rendimiento aumente proporcionalmente.

Requisitos de funcionalidad

Estos requisitos especifican la capacidad de la arquitectura para interaccionar con el entorno y satisfacer las necesidades generales o particulares de cada aplicación.

- **Ancho de palabra/precisión**. Indica el tamaño de los datos que puede manejar la arquitectura. Generalmente el ancho de palabra viene determinado por las necesidades de cada aplicación y cuanto mayor sea, mayor será la potencia y la precisión de las operaciones. La precisión, en particular, dependerá del sistema de representación utilizado por las unidades de cómputo de la arquitectura: punto fijo, coma flotante, etc.

- **Ancho de Banda**. Indica el caudal de información que es capaz de transmitir una interfaz de E/S a través de una conexión, la cual puede ser interna o externa.
- **Generalidad**. Con este parámetro expresamos la capacidad de la arquitectura para utilizar conjuntos de datos de entrada diferente al previsto inicialmente durante la fase de diseño, por ejemplo, entradas con mayor número de bits.

Requisitos de eficiencia

Estos requisitos se utilizan para establecer la relación que existe entre el rendimiento de la arquitectura y el consumo de recursos hardware. El rendimiento puede ser expresado en términos de usabilidad de los recursos, velocidad de procesamiento, tiempos de respuesta, consumo de energía y coste.

- **Uso de recursos**. Está relacionado con la capacidad de la arquitectura para hacer un buen uso de sus componentes a la hora de cumplir con los objetivos. Este requisito suele estar expresado en términos relacionados con las características del hardware: área, cantidad de recursos, ciclo de trabajo, etc.
- **Productividad (Throughput)**. Cantidad de datos que la arquitectura es capaz de procesar por unidad de tiempo. Este requisito es crítico para el funcionamiento de los sistemas en tiempo real. En el caso de que este requisito no se cumpla, la viabilidad de la arquitectura quedará en compromiso por su baja velocidad.
- **Latencia**. Especifica el tiempo que consume la arquitectura para completar el procesamiento de un dato que recibe por su entrada. Este tiempo de respuesta suele depender de múltiples factores de diseño, como por ejemplo, el modelo computacional utilizado (secuencial o paralelo), la cantidad de recursos hardware utilizados, la organización de los recursos, nivel de segmentación de las unidades de procesamiento, etc.
- **Consumo de energía**. Mide la cantidad de energía eléctrica que consume la arquitectura para llevar a cabo sus operaciones. Generalmente, el con-

sumo depende directamente del número de componentes utilizados, de la frecuencia de funcionamiento y de la productividad de la arquitectura.

- **Coste**. El coste de la arquitectura depende principalmente de la cantidad y tipo de componentes utilizados para su realización. Otro factor a considerar en el coste es el tiempo de diseño y desarrollo de la arquitectura.

Requisitos de confiabilidad

Estos requisitos miden si la calidad de los servicios básicos ofrecidos por la arquitectura es adecuada bajo ciertas condiciones de funcionamiento y durante un periodo de tiempo establecido.

- **Fiabilidad**. Es la probabilidad de que la arquitectura opere sin fallos ni degradaciones de rendimiento bajo condiciones normales de funcionamiento y durante un periodo de tiempo prolongado.
- **Robustez**. Es la cualidad de la arquitectura para seguir operando sin excesivos daños en situaciones anormales de funcionamiento: eventos inesperados, fallos de funcionamiento, distintas frecuencias de reloj, incertidumbres en las entradas, entradas con velocidad variable, etc. Los errores en una arquitectura robusta no deberán ocasionar una caída total del sistema, únicamente deberán degradar ligeramente su rendimiento.

Requisitos de portabilidad

Estos requisitos establecen la capacidad de la arquitectura para ser utilizada en diferentes ambientes. En nuestro caso estos ambientes están relacionados con la plataforma de cómputo utilizada para ejecutar las aplicaciones.

- **Adaptabilidad**. Capacidad de la arquitectura para ser adaptada, en su totalidad o parcialmente, a nuevas aplicaciones.
- **Implementabilidad**. Determina las facilidades que ofrece la arquitectura para ser implementada sobre diferentes plataformas físicas o lógicas (hardware o software).

3.3.1. Reglas de diseño

Como se ha mencionado anteriormente, el objetivo principal de este trabajo es diseñar e implementar sobre hardware reconfigurable una arquitectura de cómputo, basada en redes CNN, capaz de procesar imágenes e información de vídeo en tiempo real. Con este propósito se ha tratado de definir un conjunto de reglas de diseño que, combinando apropiadamente los requisitos mencionados en la sección anterior, permitan establecer las directrices básicas de diseño y resaltar aquellas características relevantes no consideradas en otras realizaciones.

La mayoría de los requisitos propuestos anteriormente representan cualidades y características deseables para todo tipo de arquitecturas, simplemente con su consideración sería posible mejorar su calidad y prestaciones. Sin embargo, hay que tener en cuenta que algunos de estos requisitos son más importantes que otros y que, por tanto, con el fin de garantizar la funcionalidad mínima de la arquitectura, deberán ser tratados con mayor prioridad. Una vez cubiertas las necesidades básicas de la arquitectura, el resto de requisitos, de menor prioridad, podrán ser considerados a la hora de incrementar las prestaciones de la arquitectura.

Bajo esta premisa, a continuación se estable qué requisitos de los propuestos anteriormente son necesarios para garantizar el funcionamiento de nuestra arquitectura y cuáles de ellos solamente relevantes para mejorar sus prestaciones. Se ha de tener en cuenta, además, que las prestaciones de la arquitectura dependerán de la flexibilidad del proceso de diseño y de las características de la tecnología utilizada para la realización, en particular del comportamiento de componentes elementales como: memorias, elementos de cálculo, interfaces E/S, líneas de conexión, etc.

Los requisitos estructurales, relaciones con el uso de los recursos hardware, han sido considerados imprescindibles para garantizar el almacenamiento y el procesamiento de la arquitectura y, por tanto, su funcionamiento. Para satisfacer dichos requisitos, la arquitectura debe contar al menos con una cantidad mínima de recursos hardware. Otros requisitos considerados necesarios para el funcionamiento de la arquitectura son el ancho de palabra/precisión y el ancho de banda, ya que garantizan que la arquitectura pueda recibir los datos adecuadamente y que estos puedan ser procesados con la suficiente precisión. Para adaptar la arquitectura a las aplicaciones de tiempo real se ha considerado como prioritario

satisfacer el requisito de productividad (trhroughput). De igual forma, los requisitos de fiabilidad y robustez son necesarios para garantizar el funcionamiento de la arquitectura según lo previsto por el modelo de la red CNN.

Los requisitos de modularidad, flexibilidad, escalabilidad y ampliabilidad son características importantes de la arquitectura aunque no imprescindibles para su funcionamiento. Dichos requisitos han sido tenidos en cuenta para incrementar las prestaciones de la arquitectura y facilitar la modificación y ampliación de su funcionalidad. Junto a estos requisitos, los requisitos de adaptabilidad e implementabilidad han sido considerados importantes para conseguir que el sistema pueda ser utilizado en el mayor número de aplicaciones. El requisito de sencillez ha sido prioritario en el diseño ya que facilita la comprensión de la arquitectura y simplifica su implementación y mantenimiento. Un diseño sencillo minimiza el coste de realización de la arquitectura y favorece el cumplimiento de otros requisitos imprescindibles como la robustez y la fiabilidad.

Una vez que los requisitos principales han sido satisfechos, el proceso de diseño se ha centrado en maximizar la eficiencia de la arquitectura. Para ello, se ha tratado de mejorar el compromiso entre la velocidad de procesamiento y el consumo de recursos hardware. Otros requisitos importantes como la latencia, el consumo de energía, el coste y la generalidad, también deseables en la mayoría de las arquitecturas, han sido considerados en el diseño aunque con menor prioridad. A continuación se resumen las principales reglas de diseño que se han establecido para desarrollar nuestra arquitectura:

1. Satisfacer las requisitos básicos que garanticen el funcionamiento estable y robusto de la arquitectura (requisitos estructurales, ancho de banda/palabra y precisión, fiabilidad y robustez).

2. Satisfacer los requisitos de velocidad exigidos por las aplicaciones de tiempo real.

3. Conseguir una arquitectura sencilla, modular, flexible y escalable.

4. Maximizar la eficiencia de la arquitectura en términos de área y velocidad.

Durante el diseño de la arquitectura, las reglas anteriores han sido traducidas a concreciones que, tras ser implementadas y evaluadas en un esmerado proce-

so iterativo, han dado lugar a un conjunto de reglas definitivo más preciso y equilibrado.

3.4. Paralelización de la arquitectura de cómputo

Para hacer frente a los requerimientos temporales, exigidos por las aplicaciones de tiempo real, ha sido necesario recurrir al diseño de una arquitectura de cómputo paralela, es decir, una arquitectura que explote diferentes sucesos concurrentes del modelo de cómputo mediante la ejecución simultánea o solapada de múltiples tareas sobre diferentes recursos hardware.

Almansi y Gottlieb (Almasi y Gottlieb 1989) definen el concepto de computador paralelo como: *"el conjunto de elementos de proceso, procesadores, que se comunican entre sí cooperando todos juntos en la resolución de un mismo o único problema de gran tamaño"*. A partir de esa idea, consideraremos a nuestra arquitectura de cómputo como un conjunto de componentes de cálculo interconectados entre sí, cuyo fin es resolver de forma cooperativa el modelo computacional de la red CNN, el cual previamente tendrá que ser paralelizado adecuadamente.

3.4.1. Niveles de paralelismo y granularidad

El paralelismo se dará en mayor o en menor medida en todos los niveles de abstracción de nuestra arquitectura, la organización y estructuración de los niveles ayudará a identificar y explotar el paralelismo. Para identificar el paralelismo se ha considerado que en cada nivel se desarrolla una funcionalidad diferente que puede ser dividida y paralelizada de forma independiente. La implementación del paralelismo se realizará, en función de las características y necesidades de cada nivel, combinando tanto técnicas de división funcional como de segmentación. En los niveles inferiores, la aplicación del paralelismo afectará principalmente a los requisitos de eficiencia de la arquitectura interna, mientras que en los niveles superiores afectará a la concurrencia de la arquitectura explícita de la red CNN. Los diferentes niveles de paralelismo considerados son los siguientes:

- **Paralelismo a nivel de sistema**. El paralelismo en este nivel está pensado para desarrollar sistemas neuronales complejos formados por múltiples redes

o redes de gran tamaño. El objetivo de este nivel de paralelismo es dividir el sistema neuronal en diferentes bloques y ejecutarlos simultáneamente sobre múltiples FPGAs.

- **Paralelismo a nivel de red neuronal**. Las diferentes redes neuronales que constituyen el sistema pueden ser ejecutadas en paralelo una vez resueltas las dependencias entre ellas. En este nivel se considera que las redes CNN pueden ser divididas en sub-redes y que estás pueden ejecutarse en paralelo sobre un mismo dispositivo.

- **Paralelismo a nivel de Capa/Celda**. Las redes estarán constituidas por componentes del tipo Capa y Celda que interactúan entre sí y que pueden ser ejecutados en paralelo.

- **Paralelismo a nivel de iteración**. El modelo matemático de una Celda es representado mediante un bucle, cuyas iteraciones, denominadas Etapas, pueden ser ejecutadas en paralelo tras resolverse sus dependencias.

- **Paralelismo a nivel de operador**. El procesamiento realizado por el componente Etapa viene descrito por un conjunto de operadores matemáticos que pueden ser ejecutados en paralelo. Alguno de los operadores más representativos de este nivel son, por ejemplo, la unidad de convolución o la función de activación. Desde el punto de vista del usuario, la Etapa será el componente elemental a partir del cual se construirán las redes CNN.

- **Paralelismo a nivel de operación**. Este nivel se centra en la paralelización de las operaciones matemáticas que constituyen los operadores de la Etapa. Las operaciones son llevadas a cabo mediante componentes aritméticos o lógicos que pueden ser paralelizados. Un ejemplo de este tipo de componentes es la unidad de multiplicación y acumulación (MAC), usada para desarrollar el operador de convolución.

- **Paralelismo a nivel de suboperación**. Este nivel permite explotar el paralelismo de una operación, es decir, ejecutar de forma simultánea diferentes acciones relacionadas con una única operación. Un ejemplo de paralelización a este nivel es la segmentación interna del circuito de multiplicación.

Dependiendo del nivel donde se aplique el paralelismo este quedará más o menos oculto a ojos del usuario y/o de las aplicaciones. En función de esta visibilidad distinguiremos entre paralelismo explícito y paralelismo implícito.

El paralelismo explícito está asociado a los niveles de abstracción superiores de la arquitectura (arquitectura externa) y será especificado por el diseñador de la red CNN, quien decidirá como controlarlo y explotarlo de forma óptima. Este tipo de paralelismo, se caracteriza por utilizar entidades de procesamiento similares para incrementar la velocidad de procesamiento del sistema. Por otra parte, el paralelismo implícito aparece en los niveles de abstracción inferiores (arquitectura interna) y será más abstracto y difícil de detectar por el usuario. El paralelismo implícito quedará oculto en el interior de la arquitectura y su explotación consistirá en reforzar la concurrencia interna de la microarquitectura. Cuanto menor sea el nivel de abstracción, la aplicación de este paralelismo será más compleja y exigirá un mayor conocimiento de la tecnología.

La granularidad del paralelismo está relacionado con el nivel de abstracción del sistema y aumenta o disminuye según lo hace este. El grano más pequeño (grano muy fino) ha sido asociado al nivel de suboperación, mientras que el grano mayor (grano muy grueso) ha sido asociado al nivel de paralelismo de sistema. Entre ambos extremos se distinguirá: grano fino, grano medio-fino, grano medio, grano medio-grueso y grano grueso. En la figura 3.2 se representa la correspondencia entre los niveles de abstracción de la arquitectura, el nivel de paralelismo y su granularidad.

3.5. Implementación del paralelismo

A la hora de aplicar el paralelismo se han tenido en cuenta las características de las principales arquitecturas de cómputo paralelas, las cuales han sido clasificadas en función de su paralelismo explícito e implícito.

3.5.1. Paralelismo explícito

Una de las formas más comunes de clasificar las arquitecturas de cómputo paralelas es la propuesta por M.J, Flynn (Flynn 1966). Esta clasificación (figu-

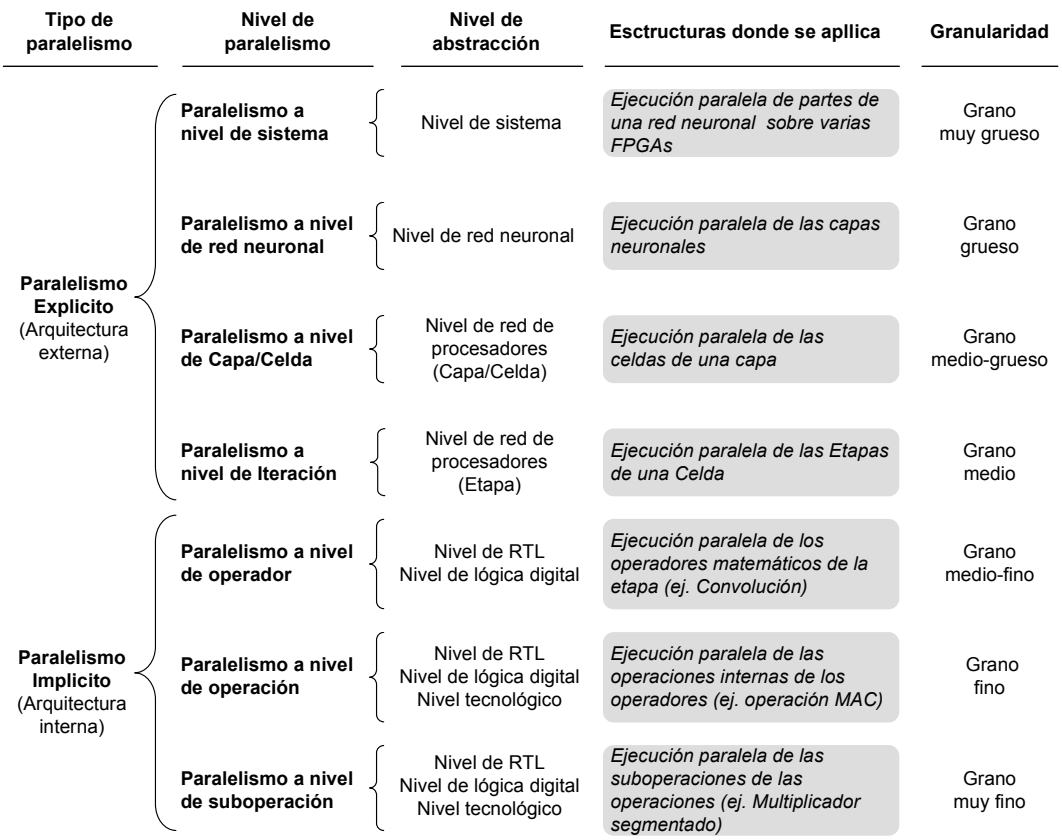


Figura 3.2: Resumen de los niveles de paralelismo aplicados a la arquitectura.

ra 3.3) considera que el paralelismo explícito de un sistema computador puede adoptar cuatro posibles configuraciones, en función de su flujo de instrucciones y su flujo de datos: SISD (*Single Instruction Single Data*), SIMD (*Single Instruction Multiple Data*), MISD (*Multiple Instruction Single Data*), MIMD (*Multiple Instruction Multiple Data*). En esta clasificación, las arquitecturas SISD son las únicas que no implementan paralelismo, ya que representan a sistemas con un único procesador (arquitectura clásica Von Neumann).

La clasificación de Flynn tiene el inconveniente, no obstante, de que algunas arquitecturas, como las vectoriales, las híbridas (MIMD-SIMD o MIMD-MISD) y otros sistemas específicos, no pueden ser clasificados de manera sencilla. El problema radica en que esta clasificación no contempla la posibilidad de que las unidades de procesamiento puedan utilizar su propia memoria local, es decir, de memoria distribuida.

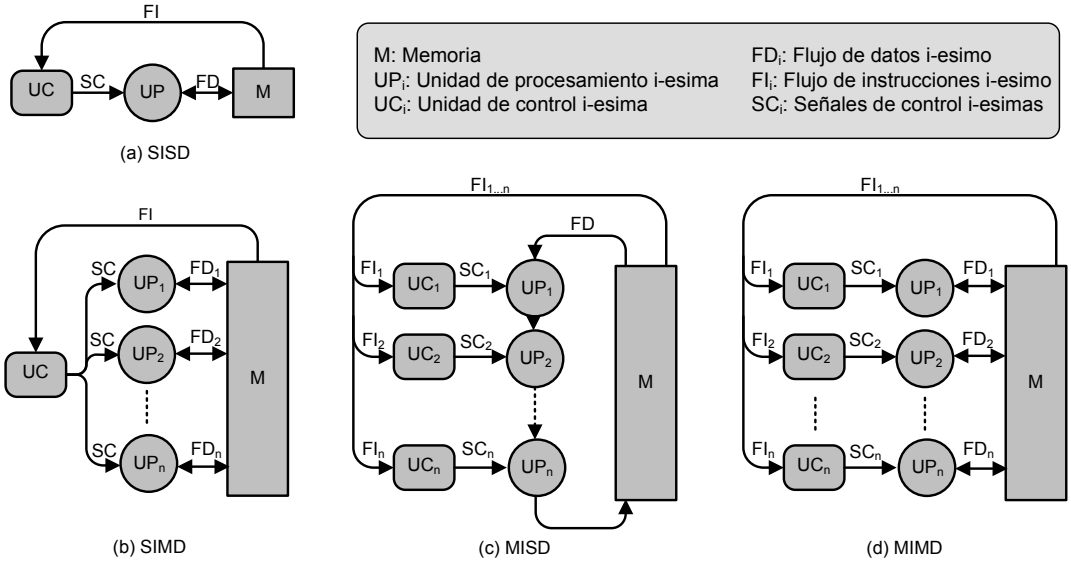


Figura 3.3: Clasificación de Flynn de las arquitecturas de cómputo.

A partir de la aproximación de K. Hwang y F.A. Briggs (Hwang y Briggs 1998) es posible ampliar la clasificación de Flynn y distinguir entre sistemas con memoria distribuida (o multicomputadores) y sistemas con memoria compartida (o multiprocesadores). En la figura 3.4 se muestra la taxonomía de Flynn ampliada, a la que se le ha añadido otros dos tipos de arquitecturas: las híbridas y las arquitecturas de propósito específico (Sriram y Bhattacharyya 2009; Pardo 2002).

La categoría MIMD puede ser dividida en tres subgrupos: multiprocesadores, multicomputadores y máquinas de flujo de datos. Dependiendo de cómo sea el modo de acceso a la memoria compartida, los multiprocesadores se clasificarán a su vez en sistemas UMA (*Uniform Memory Access*), NUMA (*Non Uniform Memory Access*) y COMA (*Cache Only Memory Access*). Por otro lado, los sistemas multicomputadores se dividen en arquitecturas de paso de mensaje y de memoria virtual compartida, en función del modo de acceso a la memoria distribuida. Finalmente, las arquitecturas de flujo de datos son máquina que ejecutan las instrucciones que demanda el flujo de datos en lugar de ejecutar secuencialmente una lista de instrucciones programada, como hacen los procesadores convencionales.

Dentro de la categoría SIMD, los procesadores matriciales representan a las arquitecturas que utilizan unidades de procesamiento idénticas para ejecutar si-

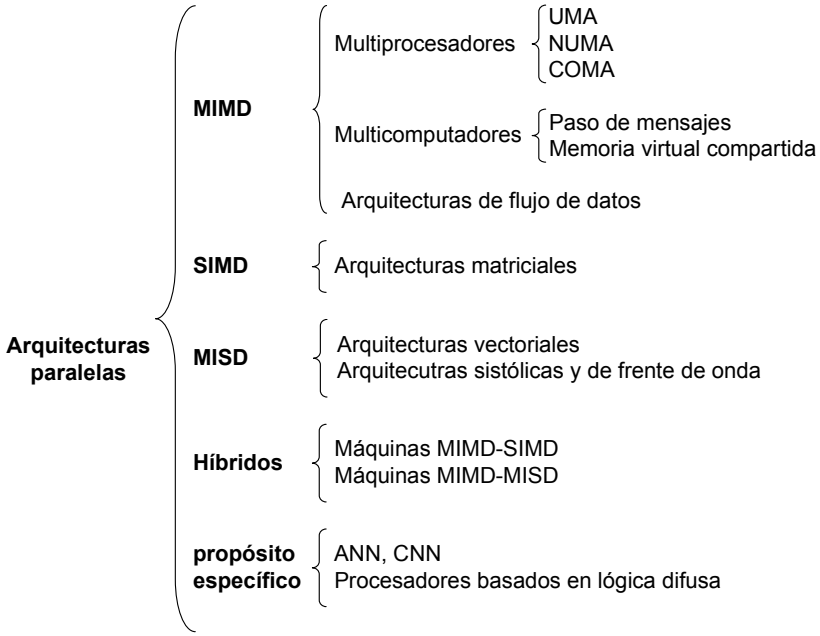


Figura 3.4: Clasificación ampliada de las arquitecturas paralelas.

multáneamente una misma instrucción sobre un conjunto diferente de datos (figura 3.3-b). En estos sistemas, todos los procesadores operan bajo una misma unidad de control, que proporciona las señales decodificadas de cada instrucción. El objetivo de estas arquitecturas es maximizar el paralelismo de los datos sin incrementar el paralelismo de las instrucciones. Estas máquinas se adaptan especialmente bien al formato particionable e independiente de las representaciones vectoriales y matriciales. Aunque su principal inconveniente es el alto coste que supone su realización hardware, ya que, generalmente, requieren tantas unidades de procesamiento como elementos tenga el vector.

Los sistemas MISD incluyen los procesadores vectoriales y las arquitecturas sistólicas y de frente de onda. Las arquitecturas vectoriales comparten ciertas similitudes con las arquitecturas matriciales ya que operan simultáneamente con todos los elementos de un vector, aunque, utilizando solamente una unidad de procesamiento vectorial. Además, dado que en la práctica es excesivamente costoso asignar una ALU a cada elemento del vector, lo que se hace es utilizar un número inferior de ALUs pero segmentadas. A pesar de que estas arquitecturas utilizan una sola unidad de control son incluidas en la categoría MISD debido a

que se considera que cada vector es un dato y que las operaciones que se realizan sobre él, a lo largo de la unidad segmentada, son las múltiples instrucciones que se aplican.

Las arquitecturas sistólicas son estructuras formadas por varios EPs (Elementos de Procesamiento) simples, con memoria local reducida, que se organizan en pipeline para construir redes de procesadores que habitualmente presentan una topología lineal o matricial. En estas arquitecturas, los EPs solamente se conectan a otros EPs adyacentes y únicamente los situados en los extremos se utilizan para establecer comunicación con el exterior. El término "sistólico" hace referencia a cómo los datos fluyen a lo largo de la red mientras son procesados usando sencillas operaciones de cálculo. Este comportamiento rítmico, tanto de los datos como de las unidades de procesamiento, representa la principal característica y ventaja de este tipo de arquitecturas. Las arquitecturas sistólicas son utilizadas fundamentalmente en sistemas donde existe una clara relación entre el ancho de banda de E/S y los requerimientos de procesamiento.

Las arquitecturas de frente de onda (Kunng 1988) son similares a las arquitecturas sistólicas, si bien su principal diferencia radica en que sus procesadores no dependen solamente de una señal de sincronización. En este caso, la transferencia de información entre los procesadores se realiza de forma asíncrona o *self-timed* (autotemporizada) usando un protocolo (*handshake*) que asegure la comunicación entre los procesadores. Esta característica permite que las arquitecturas puedan estar formadas por cauces de gran longitud sin los inconvenientes asociados al retraso de la señal de reloj (*clock skew*). La principal desventaja de estas arquitecturas, con respecto a las arquitecturas sistólicas, es la necesidad de utilizar circuitos auxiliares para realizar la sincronización y la comunicación.

Las arquitecturas híbridas tratan de integrar y combinar en la misma máquina las ventajas de los sistemas MIMD y SIMD (o MISD), con el objetivo de complementar y compensar los defectos y virtudes de ambas. Por un lado, aprovechan las ventajas de las arquitecturas MIMD para explotar el paralelismo de las instrucciones, y por otro, las características de los sistemas SIMD (o MISD) para explotar el paralelismo de los datos. La parte encargada de paralelizar las instrucciones proporciona un gran flujo de datos que es aprovechado por la parte encargada de paralelizar los datos.

Las arquitecturas específicas son creadas bajo fuertes restricciones de diseño para operar y desarrollar tareas relacionadas con aplicaciones concretas. Estas arquitecturas suelen ser utilizadas para realizar sistemas específicos (redes neuronales artificiales, procesadores basados en lógica difusa, etc.) y resolver eficientemente sus tareas de procesamiento, puesto que son diseñadas con la funcionalidad requerida por cada aplicación. También son denominadas arquitecturas VLSI ya que, generalmente, son desarrolladas usando circuitos ASIC o FPGAs.

3.5.2. Paralelismo implícito

El paralelismo implícito ha sido aplicado en los tres niveles de abstracción inferiores de nuestra arquitectura usando las dos técnicas básicas de paralelismo: la paralelización temporal (o segmentación), donde varias operaciones son ejecutadas simultáneamente sobre un cauce de etapas en cascada, y la paralelización espacial (o división funcional), donde la réplica de componentes permite ejecutar varias operaciones de forma simultánea. En función del grado de aplicación de estas dos técnicas, las unidades de procesamiento de un computador pueden ser clasificadas en cuatro tipos diferentes (figura 3.5): unidades serie escalares, unidades segmentadas, unidades supersegmentadas y unidades superescalares.

Las unidades serie escalares (figura 3.5-a) representan las unidades de procesamiento más sencillas de todas. Estas unidades no implementan ningún tipo de paralelismo ya que únicamente pueden trabajar con un solo dato a la vez, es decir, no inician el procesamiento de nuevo dato hasta que no hayan concluido el anterior.

Las unidades segmentadas (figura 3.5-b) se desarrollan a partir de un flujo de datos sobre el que se aplica una serie de etapas de procesamiento en cascada. Las etapas suelen conectarse en fila y tienen como misión realizar una tarea de procesamiento determinada. Esta técnica permite que diferentes tareas puedan ser ejecutadas simultáneamente, sobre diferentes datos, sin necesidad de esperar a que los datos anteriores hayan salido del cauce. Una vez que el cauce está completo, la velocidad de procesamiento del sistema vendrá determinada por la etapa más lenta del cauce. La técnica de segmentación ha sido utilizada en todos los niveles de abstracción de nuestra arquitectura.

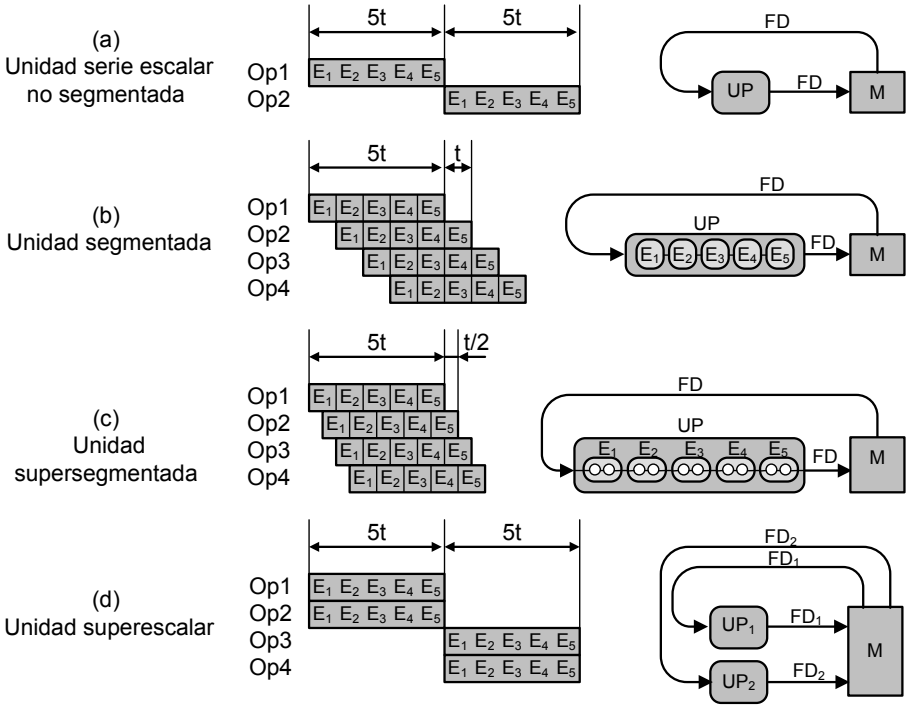


Figura 3.5: Clasificación de las unidades de procesamiento.

En las unidades supersegmentadas (figura 3.5-c) se parte del hecho de que el número de fases requeridas para ejecutar una operación puede ser mayor al número de etapas utilizadas en el cauce. Así pues, las etapas que constituyen una operación pueden ser divididas en sub-etapas que podrán ejecutarse cada cierta fracción de reloj, por ejemplo, cada medio o cuarto periodo de reloj. Al segmentar una unidad de procesamiento ya segmentada se reducirá aún más el periodo de ejecución y por consiguiente se incrementará la productividad del sistema.

En las unidades superescalares (figura 3.5-d) el incremento de velocidad se consigue ejecutando varias operaciones simultáneas sobre diferentes recursos de cómputo replicados. Como sucede en la segmentación, para poder ejecutar diferentes operaciones simultaneamente será necesario resolver con anterioridad las dependencias existentes entre las operaciones. El uso de esta técnica es compatible con las técnicas de segmentación y supersegmentación, de forma que la combinación de estas permitirá incrementar el rendimiento de las unidades de procesamiento de forma considerable. Dichas unidades contarán con microarquitecturas organizada en múltiples cauces paralelos que ejecutarán las operaciones de forma segmentada o supersegmentada.

3.6. Metodología de diseño de la arquitectura de cómputo

El diseño de una arquitectura de cómputo es un proceso complejo que requiere la consideración de múltiples factores, uno de los más importantes es la concurrencia del hardware. Cuando una arquitectura hardware es diseñada para ejecutar múltiples tareas de procesamiento simultáneas será necesario considerar la sincronización y la comunicación de los componentes que la constituyen. Otro aspecto importante a tener en cuenta es cómo llevar a cabo el particionamiento del algoritmo que implementa la arquitectura paralela. Un particionamiento inteligente repercutirá en un mayor aprovechamiento de los recursos internos del hardware y facilitará la comunicación y sincronización de sus componentes. No obstante, realizar un buen particionamiento no es sencillo pues requiere un conocimiento profundo del problema y una estimación previa del resultado de la arquitectura.

El proceso de diseño de una arquitectura de cómputo es una tarea creativa que requiere grandes dosis de experiencia e intuición. A partir del análisis del problema inicial es conveniente establecer un conjunto de reglas de partida que permitan sistematizar el proceso y facilitar su realización. En este sentido, a continuación se indican las etapas que se han tenido en cuenta a la hora de abordar el diseño y la realización hardware de nuestra arquitectura de cómputo (figura 3.6):

1. *Determinar el modelo discreto de la red CNN.*
2. *Especificar el algoritmo de cómputo.*
3. *Paralelizar el algoritmo.*
4. *Implementar la arquitectura.*
5. *Evaluar y medir el rendimiento.*

- **Determinar el modelo discreto de la red CNN**. El proceso de diseño de nuestra arquitectura se inicia con la especificación del modelo de cómputo de la red CNN a implementar. Dicho modelo es obtenido a partir de la

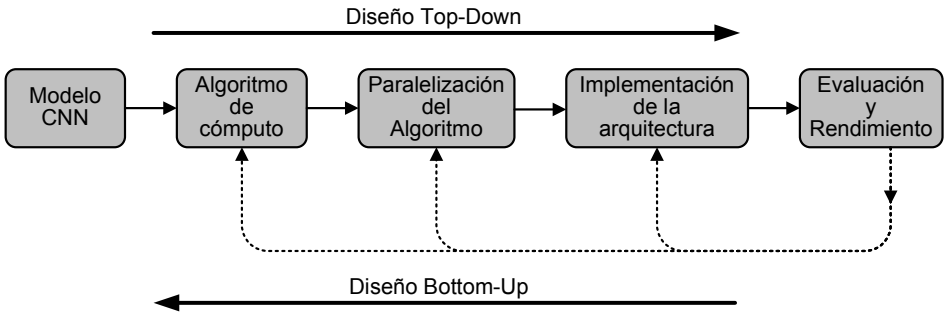


Figura 3.6: Proceso de diseño de la arquitectura de cómputo.

discretización del modelo continuo de la red CNN usando diferentes métodos de aproximación. El objetivo de esta fase es conseguir diferentes modelos discretos aproximados para compararlos entre sí y determinar cuál de ellos ofrece mayor precisión y eficiencia computacional.

- **Especificar el algoritmo de cómputo**. El algoritmo de cómputo, basado en el modelo obtenido en la fase anterior, deberá especificar de forma clara y precisa las etapas y operaciones que definen a la arquitectura. En la especificación, el algoritmo deberá ser adaptado a las características del hardware y sus cálculos expresados de forma eficiente. Para simplificar la etapa de diseño siguiente, y favorecer la paralelización y el aprovechamiento de los recursos hardware, es conveniente que el algoritmo sea especificado teniendo en cuenta algunas de las arquitectura explícitas de la sección 3.5.1, eligiendo aquella que mejor se adapte a las características del algoritmo, del hardware y de las aplicaciones.

- **Paralelizar el algoritmo**. La paralelización consistirá en dividir el algoritmo en unidades de cómputo de menor tamaño con el fin de extraer su máxima concurrencia. En nuestro caso, la paralelización se ha dividido en dos fases: la descomposición del algoritmo en tareas y la agrupación de tareas en procesos. En la fase de descomposición, el algoritmo es divido en unidades de cómputo de pequeño tamaño (tareas) que incluirán parte de los cálculos y de la estructura de datos del algoritmo. En la fase de agrupación, las tareas son combinadas en bloques de mayor tamaño (procesos) con el objetivo de adaptar la descomposición a los requisitos del hardware y simplificar la implementación.

- **Implementar la arquitectura**. En esta fase, la arquitectura es concretada en función del particionamiento y el agrupamiento realizados en el etapa anterior. La mayoría de las prestaciones de la arquitectura se determinan en esta fase y dependerán tanto de la tecnología utilizada para la realización como de la habilidad del diseñador para la implementación. Tras conocer la arquitectura explícita y los detalles de la tecnología, los procesos definidos en la etapa anterior son implementados en hardware aprovechando las técnicas del paralelismo implícito (sección 3.5.2), con el fin de mejorar las prestaciones de la arquitectura interna.

- **Evaluar y medir el rendimiento**. El paso siguiente en el proceso de diseño es evaluar el funcionamiento y el rendimiento de la arquitectura. Si las prestaciones obtenidas no alcanzan los requisitos deseados, la arquitectura deberá ser revisada volviendo a etapas anteriores. Se retrocederán más pasos hacia atrás cuanto mayor sea la desviación entre los requisitos previstos y los realmente alcanzados.

El desarrollo de la arquitectura hardware se ha realizado combinando simultáneamente técnicas de diseño *Top-down* y *Bottom-up*. El diseño *Top-down* parte de las características y requisitos del sistema y se centra en el conocimiento, la organización y planificación de la jerarquía de la arquitectura. En este flujo de diseño, los componentes de mayor nivel son definidos y refinados, cada vez con mayor detalle, hasta alcanzar los requisitos deseados. Por otra parte, el diseño *Botton-up* hace hincapié en el desarrollo eficiente de la arquitectura implícita. Los componentes de alto nivel son diseñados, desde su concepción, considerando los detalles de la tecnología usada, con el fin de estimar sus capacidades y el nivel de cumplimiento de los requisitos exigidos.

3.7. Discretización del modelo de la red CNN

El proceso de diseño de la arquitectura comienza con la especificación del modelo de cómputo que implementa la arquitectura, y que ha sido obtenido a partir de la discretización del modelo continuo de la red CNN. Previamente, diferentes modelos discretos, obtenidos mediante distintos métodos aproximados,

han sido analizados y comparados entre sí con el fin de comprobar sus prestaciones y seleccionar el más adecuado para la implementación sobre FPGA.

Para elegir el modelo discreto de la red CNN se han tenido en cuenta las características internas de las FPGAs, en particular las de los recursos específicos que dan soporte a las funciones básicas de procesamiento DSP (Digital Signal Processing): Bloques de memoria, multiplicadores embebidos, lógica de acarreo rápida, etc. Estos elementos permiten incrementar la flexibilidad y capacidad de cómputo de las FPGAs y facilitan el mapeo de las estructuras de procesamiento de la arquitectura. No obstante, hay que considerar que dichos recursos son limitados y que tienen que ser aprovechados y reutilizados tanto como sea posible con el fin de maximizar la eficiencia de la arquitectura. Por esta razón, los métodos de discretización utilizados tienen como objetivo conseguir una buena aproximación al modelo continuo de referencia y aprovechar los recursos internos de la FPGA de forma eficiente.

El proceso de discretización parte de la transformada de Laplace del modelo continuo de la red CNN. A partir de esta transformación, se proponen cuatro métodos de discretización diferentes que proporcionan distintos modelos aproximados con características y prestaciones diferentes, tres de ellos de primer orden y el cuarto de segundo orden. La primera aproximación se caracteriza por un comportamiento predictivo, la segunda por un comportamiento invariante en el tiempo (sin retrasos ni adelantos en su respuesta) y la tercera y cuarta por un comportamiento retrasado en el tiempo. Los cuatro modelos aproximados han sido simulados y comparados con el modelo continuo de la red CNN con el fin de comprobar su precisión y analizar sus diferencias.

3.7.1. Función de transferencia del modelo continuo de la red CNN

La respuesta dinámica del modelo continuo de la red CNN viene determinada por la ecuación de estado 3.1 y la función de salida 3.2.

$$\frac{d}{dt}x_{ij}(t) = -x_{ij}(t) + \sum_{k,l \in Nr(ij)} A_{kl}y_{kl}(t) + \sum_{k,l \in Nr(ij)} B_{kl}u_{kl} + I_{ij} \tag{3.1}$$

$$y_{ij}(t) = \frac{1}{2}(|x_{ij}(t)+1| - |x_{ij}(t)-1|) \tag{3.2}$$

El modelo continuo de la red CNN puede ser transformado al dominio de Laplace sustituyendo la expresión 3.3 en la ecuación de estado 3.1. El resultado de esta transformación, ecuación 3.4, puede ser representando gráficamente mediante el diagrama de bloques de la figura 3.7.

$$\frac{d}{dt}x(t) = X \cdot S \tag{3.3}$$

$$X_{ij} = \left(\sum_{k,l \in Nr(ij)} A_{kl} Y_{kl} + \sum_{k,l \in Nr(ij)} B_{kl} U_{kl} + I_{ij} \right) \frac{1}{S+1} \tag{3.4}$$

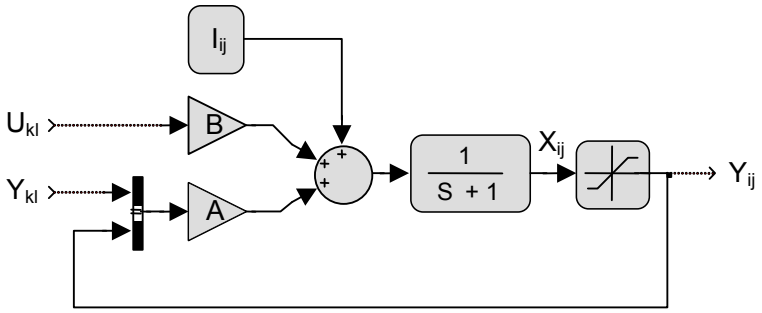


Figura 3.7: Diagrama de bloques de la celda CNN estándar en el dominio de la Laplace.

La función de transferencia de la ecuación del sistema en el dominio de la Laplace viene determinada por la ecuación 3.5:

$$H(S) = \frac{1}{S+1} \tag{3.5}$$

$$X_{ij} = \left(\sum_{k,l \in Nr(ij)} A_{kl} Y_{kl} + \sum_{k,l \in Nr(ij)} B_{kl} U_{kl} + I_{ij} \right) H(S) \tag{3.6}$$

3.7.2. Aproximaciones discretas del modelo de la red CNN

El modelo discreto de la red CNN ha sido obtenido transformando la ecuación de estado 3.6 al dominio discreto Z. Dado que un sistema real requiere una transformada Z racional, las transformaciones utilizadas darán como resultado modelos discretos aproximados. Alguno de los métodos más utilizados son: el método de

Euler, la Transformación Invariante a la Respuesta (TIR), la Transformada Z Pareada (TZP), la Transformada mediante Algoritmos Diferenciales (TAD), la Transformada mediante Algoritmos de Integración numérica (TAI), etc.

Con el fin de minimizar el coste computacional de las aproximaciones, los métodos de discretización tendrán que ser simples y utilizar el menor orden de integración posible. Bajo esta consideración, los métodos elegidos han sido: el método directo de Euler de primer orden (que denominaremos aproximación de Euler), la transformada invariante al impulso de primer orden (aproximación TIR), el algoritmo hacia atrás de primer orden (aproximación TAD) y el método de integración numérica del trapecio de segundo orden (aproximación TAI).

Aproximación por el método de Euler (método directo de Euler)

Con este método, el modelo discreto de la red CNN es obtenido a partir del desarrollo de Taylor de la ecuación 3.6. En el caso de utilizar solamente el primer término de la factorización de Taylor (aproximación de primer orden), el modelo aproximado obtenido es el más sencillo de todo y se denomina aproximación directa de Euler. A partir de este método se deduce que una función continua $x(t)$ puede ser aproximada, en un instante de tiempo posterior $x(t+h)$, por la expresión 3.7.

$$x(t+h) \approx x(t) + \frac{\mathrm{d}x(t)}{\mathrm{d}t}h \tag{3.7}$$

Donde h representa un incremento de tiempo pequeño, que en nuestro caso corresponde con la constante de muestreo del sistema. Según esta expresión, los modelos obtenidos por este método tendrán carácter predictivo (respuesta adelantada) y, por tanto, su comportamiento podrá ser inestable.

Aplicando la transformada 3.3 a la expresión 3.7 se obtiene una aproximación equivalente en el dominio discreto Z, ecuación 3.8:

$$S \approx \frac{Z-1}{h} \tag{3.8}$$

Sustituyendo a su vez la expresión anterior en la ecuación 3.5 se obtiene la función de transferencia de la celda CNN en el dominio discreto Z, ecuación 3.9:

$$H(Z) = \frac{h}{Z + (h-1)} \tag{3.9}$$

Finalmente, la sustitución de la función de transferencia 3.9 en la ecuación continua 3.6 da como resultado la aproximación de Euler del modelo de la red CNN, ecuación 3.10:

$$X_{ij}[k] = (1-h)X_{ij}[k-1] + h\left(\sum_{k,l\in Nr(ij)} A_{kl}Y_{kl}[k-1] + \sum_{k,l\in Nr(ij)} B_{kl}U_{kl} + I_{ij}\right) \tag{3.10}$$

Aproximación mediante la Transformación Invariante a la Respuesta (TIR, transformación invariante al impulso de primer orden)

Este método consiste en aplicar la transformada Z a un sistema genérico del mismo orden que el sistema continuo y, a continuación, comparar ambos sistemas ante un mismo estimulo de entrada típico (impulso, escalón, o rampa). El modelo discreto se obtendrá identificando coeficientes en ambos sistemas, una vez que sus secuencias de salida, muestreadas en intervalos de tiempo h, alcanzan el régimen permanente. Esta clase de aproximaciones se caracteriza por presentar una respuesta ideal e invariante, que se superpone a la del modelo continuo. Las transformaciones más utilizadas en estas aproximaciones son: la transformación invariante al impulso, la transformación invariante al escalón y la transformación invariante a la rampa. En nuestro caso se ha elegido la transformación invariante al impulso debido a su mayor sencillez y menor coste computacional.

La discretización del modelo de la red CNN parte de la transformada Z de un sistema de primer orden genérico sometido a la entrada impulso:

$$H(S) = \frac{A}{S+a} \quad \Rightarrow \quad H(Z) = K\frac{AZ}{Z-e^{-ah}} \tag{3.11}$$

Donde K es una constante que depende de las condiciones de contorno del sistema y que se calcula igualando la respuesta en régimen permanente de los sistemas continuo y discreto ante un escalón unitario. Una vez calculada la constante K, se procede a identificar los coeficientes de las expresiones 3.5 y 3.11 que determinan la función de transferencia de la celda en el dominio discreto Z (ecuación 3.12):

$$H(Z) = \frac{1-e^{-h}}{1-e^{-h}Z^{-1}} \tag{3.12}$$

Expresión que puede sustituirse en la ecuación 3.6 para obtener la aproximación TIR del modelo de la red CNN, ecuación 3.13:

$$X_{ij}[k] = e^{-h}X_{ij}[k-1]+(1-e^{-h})\left(\sum_{k,l\in Nr(ij)} A_{kl}Y_{kl}[k-1] + \sum_{k,l\in Nr(ij)} B_{kl}U_{kl} + I_{ij}\right) \tag{3.13}$$

Transformación mediante Algoritmos Diferenciales (TAD, método hacia atrás)

En este caso, la discretización del modelo se obtiene sustituyendo el operador derivada de la ecuación de estado 3.1 por una aproximación en diferencias equivalente, muestreada en intervalos de tiempo constante (h). El carácter de la aproximación (predictivo, retrasado o invariante) depende del método utilizado para estimar el operador derivada, que se clasifican en: métodos hacia atrás, métodos centrales o métodos hacia adelante.

El método hacia adelante de primer orden es un método de carácter predictivo que únicamente utiliza muestras futuras para estimar el operador derivada. Con este método se obtiene el mismo resultado que con el método de Euler y, por tanto, no ha sido utilizado. El método central utiliza tanto muestras futuras como pasadas y, a pesar de que suele ofrecer mejores resultados, tampoco ha sido utilizado debido a su mayor orden y coste computacional. Por último, el método hacia atrás, de primer orden, es un método que emplea muestras pasadas y que genera modelos con respuestas retrasadas con respecto a las del modelo continuo. Se ha seleccionado este método debido a la estabilidad que dicho retraso proporciona a la respuesta de los modelos y a la posibilidad de comparar ésta con la obtenida por los métodos Euler y TIR, de carácter predictivo e invariante respectivamente.

El método TAD hacia atrás de primer orden se basa en la sustitución del operador derivada por la aproximación 3.14.

$$\frac{\mathrm{d}x}{\mathrm{d}t} \approx \frac{x(t)-x(t-h)}{h} \quad \Rightarrow \quad S \approx \frac{1-Z^{-1}}{h} \tag{3.14}$$

De manera similar a como se hizo en el método de Euler, la expresión anterior puede ser sustituida en la ecuación 3.5 para obtener la versión discreta de la

función de transferencia de la celda CNN, ecuación 3.15.

$$H(Z) = \frac{h}{(1+h) - Z^{-1}} \tag{3.15}$$

Que, una vez sustituida en la ecuación de estado 3.6, permite obtener la aproximación TAD del modelo de la red CNN, expresión 3.16.

$$X_{ij}[k] = \frac{1}{(1+h)} \left[X_{ij}[k-1] + h \left(\sum_{k,l \in Nr(ij)} A_{kl} Y_{kl}[k-1] + \sum_{k,l \in Nr(ij)} B_{kl} U_{kl} + I_{ij} \right) \right] \tag{3.16}$$

Transformación mediante Algoritmos de Integración numérica (TAI, método del trapecio)

De forma similar a como se hace en los métodos TAD, los métodos TAI aproximan la ecuación diferencial de un modelo continuo mediante la aplicación de algoritmos de integración numérica. Algunos de los métodos más utilizados en la discretización de sistemas reales son: el método del rectángulo, el método del trapecio (también conocido como bilineal o Tustin), los algoritmos de Adams, Simpsom, Tick, etc. Desde el punto de vista de la realización hardware, el método más interesante es el del rectángulo debido a su sencillez y menor orden de integración. No obstante, este método ha sido descartado debido a que proporciona el mismo resultado que el método de Euler. En su lugar se ha utilizado el método del trapecio, que se caracteriza por ser de segundo orden y ofrecer una respuesta retrasada en el tiempo.

A partir del algoritmo del trapecio es posible obtener la siguiente relación:

$$\int_{h-1}^{h} x(t)\mathrm{d}t \approx \frac{x(t) + x(t-h)}{2}\, h \quad \Rightarrow \quad S \approx \frac{2}{h}\, \frac{1 - Z^{-1}}{1 + Z^{-1}} \tag{3.17}$$

Expresión que se ha utilizado para discretizar la función de estado de la celda (ecuación 3.5) y transformarla en la expresión 3.18:

$$H(Z) = \frac{h + hZ^{-1}}{(2+h) + (h-2)Z^{-1}} \tag{3.18}$$

La cual, una vez sustituida en la ecuación de estado 3.6, da como resultado la apro-

ximación del modelo de la red CNN por el método del trapecio (ecuación 3.19):

$$X_{ij}[k] = \frac{1}{2+h}\Bigg[(2-h)X_{ij}[k-1] + h\Bigg(\sum_{k,l\in Nr(ij)} A_{kl}Y_{kl}[k-1] + \\ + \sum_{k,l\in Nr(ij)} A_{kl}Y_{kl}[k-2]\Bigg) + 2h\Bigg(\sum_{k,l\in Nr(ij)} B_{kl}U_{kl} + I_{ij}\Bigg)\Bigg] \tag{3.19}$$

3.7.3. Simulación de los modelos

El modelo continuo de la red CNN y las aproximaciones discretas obtenidas en la sección anterior han sido simulados con el objeto de analizar su comportamiento y compararlo. Con el fin de poder estudiar los modelos bajo diferentes condiciones de funcionamiento se han planteado dos tipos de simulaciones diferentes: una en lazo abierto y otro en lazo cerrado. La simulación en lazo abierto tiene como objetivo analizar la respuesta individual de las celdas frente a estímulos de entrada típicos: el impulso y el escalón. Mientras que la simulación en lazo cerrado trata el análisis del comportamiento de los modelos bajo condiciones normales de funcionamiento, es decir, cuando todas las celdas están interconectadas entre sí e influyen unas sobre otras.

En la simulación en lazo abierto, la celda genérica de la red CNN (C_{ij}) ha sido aislada y su entrada de realimentación puesta a valor cero ($Y_{ij} = 0$). En la simulación en lazo cerrado, todas las celdas se interconectan localmente y reciben la influencia de todas sus entradas, incluyendo la de realimentación. En esta simulación, los modelos han sido configurados para llevar a cabo tres tipos de procesamientos de imágenes básicos: difusión de una imagen, detección de bordes y umbralización.

Todas las simulaciones realizadas, tanto en lazo abierto como en lazo cerrado, han sido llevadas a cabo utilizando un PC convencional y el entorno de desarrollo de MATLAB.

Simulación en lazo abierto

La simulación en lazo abierto de los modelos se han realizado empleando dos tipos de estímulos de entrada diferentes: la señal impulso unitario y la señal

escalón unitario. En esta simulación, las celdas de los modelos son aisladas y el operador de realimentación de su plantilla es puesto a cero ($A = 0$). Conforme los estímulos de entrada son aplicados, las secuencias de entrada y salida de la celda son almacenadas en memoria para su posterior representación gráfica. El periodo de muestreo empleado en la simulación se ha fijado a un valor constante $h = 0.5$, mitad del valor de la constante de tiempo del sistema ($T = 1$).

La figura 3.8 muestra los resultados de la simulación de los modelos en lazo abierto. Como se observa, la respuesta que más se aproxima a la del modelo continuo de referencia, para los dos tipos de estímulos propuestos, es la correspondiente a la aproximación TIR, que como era de esperar no presenta adelanto ni retraso en su respuesta. Por contra, la respuesta que más se desvía del objetivo previsto, con un cierto adelanto, debido a su carácter predictivo, es la aproximación de Euler. Por otra parte, los modelos TAD y TAI presentan retraso en la respuesta, y entre ambos, la aproximación TAI converge más rápidamente debido a su mayor orden de integración.

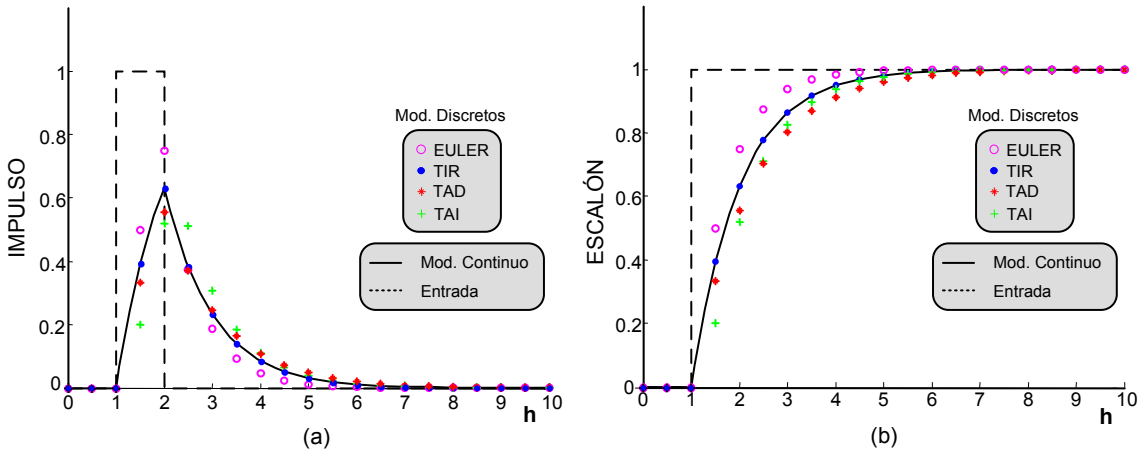


Figura 3.8: Respuesta de la celda genérica de los modelos en lazo abierto. (a) respuesta al impulso. (b) respuesta al escalón.

Simulación en lazo cerrado

La simulación en lazo cerrado tiene como objetivo analizar el comportamiento de los modelos bajo condiciones normales de funcionamiento, es decir, cuando todas las celdas de la red están interconectadas localmente y sometidas a la

influencia de todas sus entradas (U,Y,I), incluyendo las de realimentación. Con el fin de realizar un análisis más significativo, los modelos han sido simulados resolviendo tres tipos de procesamiento de imágenes diferentes: difusión, detección de bordes y umbralización, aplicados sobre diferentes imágenes de test.

Para realizar las simulaciones se han tenido en cuenta las siguientes consideraciones: radio de vecindad de las celdas igual a 1 (ventana 3×3), constante de tiempo del modelo continuo $T = 1$, diez iteraciones por simulación para el caso de los modelos discreto (paso fijo $h = 1$), operaciones con aritmética en punto fijo (18 bits) e imágenes de tamaño 200×150 píxeles codificadas con 256 niveles de grises (normalizados al intervalo $[-1, 1]$). Para comparar la respuesta de los modelos discretos con la del modelo continuo se ha utilizado el coeficiente de correlación bidimensional de las imágenes resultantes, que es calculado en cada iteración usando la siguiente expresión 3.20.

$$corr = \frac{\sum_i \sum_j (x_{ij} - \overline{x})(y_{ij} - \overline{y})}{\sqrt{\left(\sum_i \sum_j (x_{ij} - \overline{x})^2\right)\left(\sum_i \sum_j (y_{ij} - \overline{y})^2\right)}} \tag{3.20}$$

Donde x_{ij} es el pixel (i, j) de la imagen obtenida por la aproximación discreta, y_{ij} el píxel (i, j) obtenido por el modelo continuo de referencia y $\overline{x}$ e $\overline{y}$ la media de los píxeles de la imagen resultante del modelo discreto y del modelo continuo respectivamente.

El cuadro 3.1 muestra las plantillas utilizadas para realizar los diferentes procesamientos de imágenes propuestos. Dichas plantillas han sido codificadas en complemento a dos usando aritmética en punto fijo con 18 bits de precisión.

Las figuras 3.9, 3.10 y 3.11 resumen los resultados de las simulaciones tras configurar los distintos modelos con dichas plantillas. En todos los casos, se ha comprobado que los resultados obtenidos son independientes de las imágenes de test aplicada y que, por tanto, pueden ser considerados representativos de cada tipo de procesamiento. Las figuras incluyen: la imagen de test usada para realizar el procesamiento (figura a), las imágenes resultantes obtenidas por el modelo continuo y por el modelo discreto más aproximado (figura b y c, respectivamente), una gráfica con la evolución de algunos píxeles de las imágenes de salida (figura d) y una tabla con los coeficientes de correlación entre las imágenes de salida de las aproximaciones discretas y del modelo continuo (figura e).

Cuadro 3.1: Plantillas utilizadas en la simulación en lazo cerrado.

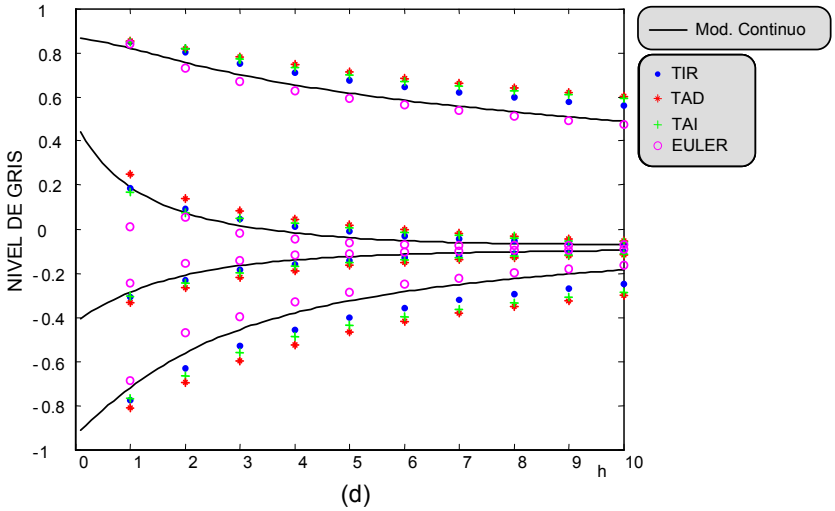
Algoritmo	Operador A	Operador B	Bias I
Difusión	$\begin{pmatrix} 0 & 1/2 & 0 \\ 1/12 & 1/1.5 & 1/12 \\ 0 & 1/12 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	0
Det. de Bordes	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{pmatrix}$	0.8
Umbralización	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$	0

Para el caso de la difusión, la imagen de salida aparece difuminada debido a que el nivel de brillo de todos los píxeles tienden asintoticamente a un valor comprendido entre $(-1, 1)$, por debajo del nivel de saturación. Por contra, tanto en la detección de bordes como en la umbralización, los píxeles de salida tienden a la saturación y, por tanto, son truncados a $+1$ o -1 por la función de activación. La diferencia entre estos dos tipos de procesamiento radica en la evolución dinámica de sus respuestas, más rápida para el caso de la umbralización que para el caso de la detección de bordes.

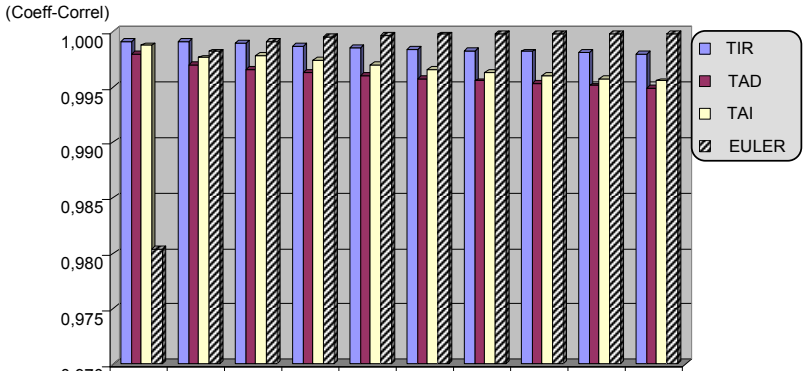
Los procesamientos de imágenes seleccionados son representativos de los dos tipos de salida que la red CNN puede proporcionar: salida no saturada (definida por píxeles con un determinado nivel de gris) y salida saturada o binaria (donde los píxeles solo pueden ser blancos o negros), en este último caso, los dos procesamientos propuestos ofrecen diferente velocidad de convergencia.

La figura 3.9 muestra los resultados proporcionados por el procesamiento de difusión. Como se observa en la figura 3.9-e, en la primera iteración, la aproximación TIR presenta el mejor comportamiento de todos, mientras que el modelo de Euler destaca por ser el menos preciso. Esta situación es transitoria y a partir de la tercera iteración la aproximación de Euler pasa a ser el modelo más aproximado hasta el final de la simulación. Otro importante resultado, que puede ser

(a) (b) (c)



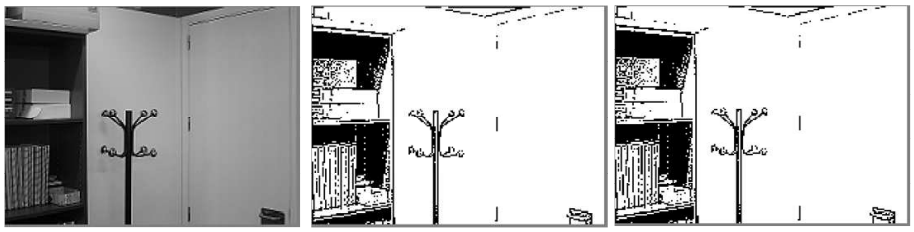
(d)



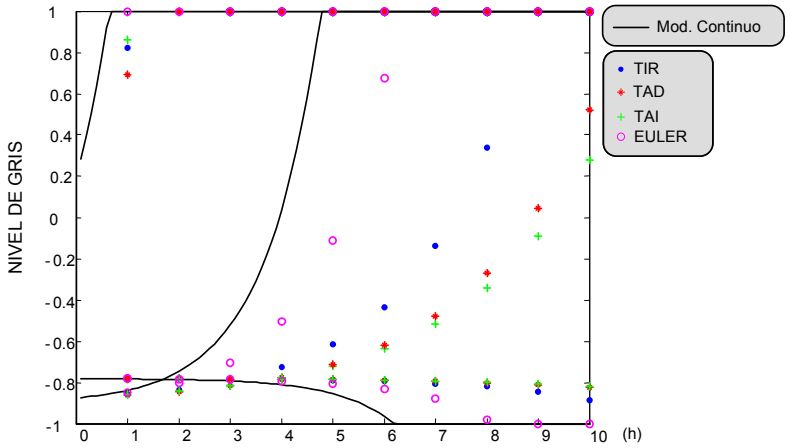
	1	2	3	4	5	6	7	8	9	10
TIR	0,9991	0,9991	0,9989	0,9987	0,9985	0,9984	0,9983	0,9982	0,9981	0,9980
TAD	0,9979	0,9970	0,9965	0,9962	0,9960	0,9957	0,9955	0,9953	0,9951	0,9949
TAI	0,9987	0,9976	0,9978	0,9974	0,9970	0,9966	0,9963	0,9960	0,9958	0,9955
EULER	0,9804	0,9982	0,9991	0,9995	0,9997	0,9997	0,9998	0,9998	0,9998	0,9999

(e)

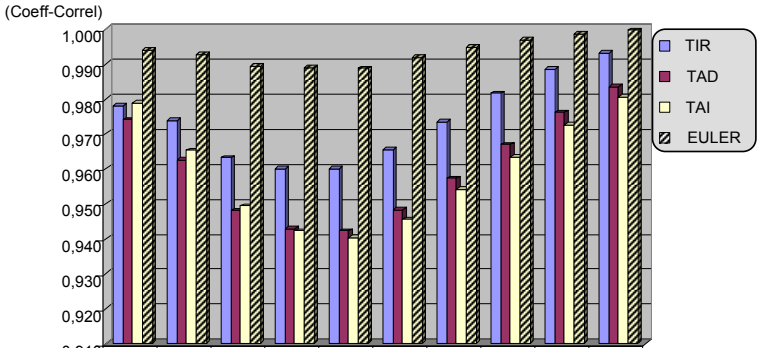
Figura 3.9: Simulación del procesamiento de difusión. (a) Imagen de test. (b) Respuesta del modelo continuo. (c) Respuesta del modelo de Euler. (d) Evolución de los píxeles. (e) Tabla de coeficientes de correlación.



(a) (b) (c)



(d)

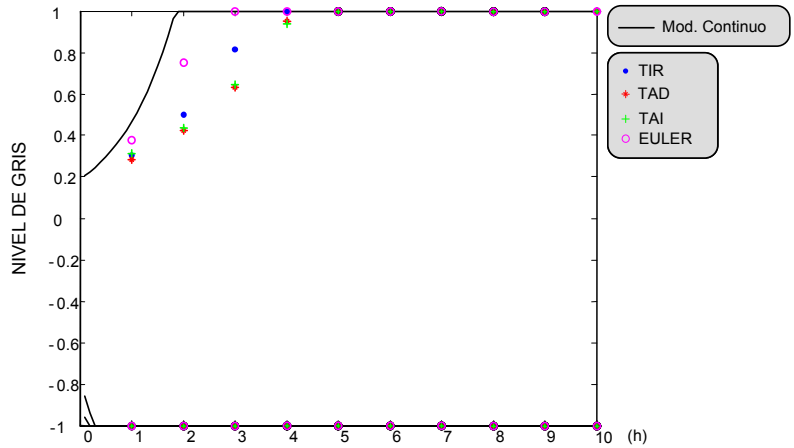


	1	2	3	4	5	6	7	8	9	10
TIR	0,9780	0,9739	0,9631	0,9601	0,9601	0,9654	0,9734	0,9816	0,9885	0,9932
TAD	0,9741	0,9625	0,9480	0,9429	0,9421	0,9483	0,9573	0,9669	0,9761	0,9836
TAI	0,9790	0,9652	0,9493	0,9422	0,9402	0,9455	0,9542	0,9635	0,9727	0,9805
EULER	0,9939	0,9927	0,9894	0,989	0,9888	0,9918	0,9949	0,9971	0,9988	0,9997

(e)

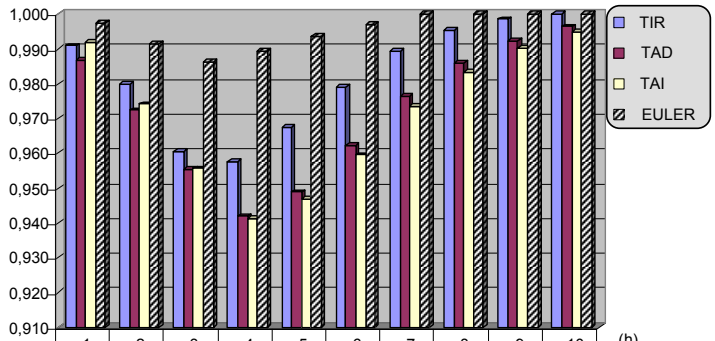
Figura 3.10: Simulación del procesamiento de detección de bordes. (a) Imagen de test. (b) Respuesta del modelo continuo. (c) Respuesta del modelo de Euler. (d) Evolución de los píxeles. (e) Tabla de coeficientes de correlación.

(a) (b) (c)



(d)

(Coeff-Correl)



	1	2	3	4	5	6	7	8	9	10
TIR	0,990	0,979	0,960	0,957	0,967	0,979	0,989	0,995	0,998	1,000
TAD	0,986	0,972	0,955	0,941	0,948	0,962	0,976	0,985	0,992	0,996
TAI	0,991	0,974	0,955	0,941	0,946	0,959	0,973	0,983	0,990	0,994
EULER	0,998	0,992	0,986	0,989	0,993	0,997	1	1	1	1

(e)

Figura 3.11: Simulación del procesamiento de umbralización. (a) Imagen de test. (b) Respuesta del modelo continuo. (c) Respuesta del modelo de Euler. (d) Evolución de los píxeles. (e) Tabla de coeficientes de correlación.

observado en la figura 3.9-e, es que mientras que el modelo de Euler mejora su respuesta cada iteración, el resto ralentiza su convergencia.

Las figuras 3.10 y 3.11 muestran respectivamente, los resultados obtenidos por los procesamientos de detección de borde y umbralización para la misma imagen de entrada. En ambos casos, hay que destacar que las imágenes obtenidas son en blanco y negro y que la aproximación de Euler mejora claramente los resultados de las restantes aproximaciones.

3.7.4. Evaluación de los modelo discreto de la red CNN

Tras analizar las simulaciones se ha podido comprobar que las aproximaciones TDA y TAI presentan un cierto retraso en sus respuestas y que este, a pesar de ser más estables y uniformes es menos preciso que la de los modelos TIR y Euler. Por otra parte, la aproximación TIR muestra un comportamiento ideal en lazo abierto, pero, sin embargo, en lazo cerrado presenta menor precisión que la obtenida por el modelo de Euler. Finalmente, el modelo de Euler tiene un mal comportamiento en lazo abierto, debido a su carácter predictivo, aunque se consolida como la mejor aproximación de todas en lazo cerrado.

Desde el punto de vista de la eficiencia en el uso de los recursos hardware, una vez desarrolladas las ecuaciones de los modelos, se ha podido comprobar que las aproximaciones TDA, TIR y Euler necesitan la misma cantidad de elementos de cálculo, mientras que la aproximación TAI, al ser de segundo orden, requiere mayor cantidad. Entre las aproximaciones de primer orden, la aproximación de Euler será la opción más interesante debido a que además ofrece un comportamiento más preciso en lazo cerrado.

Para minimizar el consumo de recursos hardware y facilitar la implementación de la arquitectura, el modelo de Euler (ecuación 3.10) ha sido modificado acotando su variable de estado al rango de la salida ($X_{ij} = Y_{ij}$), de forma similar a como se hace en el modelo de rango completo acotado (FSR). Además, la constante de muestreo del sistema h ha sido incluida en los coeficientes de la plantilla para eliminar el primer termino de la ecuación 3.10. Las simplificaciones introducidas en el modelo han dado lugar a las expresiones 3.21 y 3.22, que han sido utilizadas como punto de partida para desarrollar la arquitectura de cómputo del sistema.

$$Y_{ij}[k] = \sum_{k,l \in Nr(ij)} A_{kl} Y_{kl}[k-1] + \sum_{k,l \in Nr(ij)} B_{kl} U_{kl} + I_{ij} \tag{3.21}$$

$$A = \begin{pmatrix} ha_{-1,-1} & ha_{-1,0} & ha_{-1,1} \\ ha_{0,-1} & 1 - h + ha_{0,0} & ha_{0,1} \\ ha_{1,-1} & ha_{1,0} & ha_{1,1} \end{pmatrix}, B = \begin{pmatrix} hb_{-1,-1} & hb_{-1,0} & hb_{-1,1} \\ hb_{0,-1} & hb_{0,0} & hb_{0,1} \\ hb_{1,-1} & hb_{1,0} & hb_{1,1} \end{pmatrix} \tag{3.22}$$

Capítulo 4

Arquitecturas Carthago y Carthagonova

En este capítulo se describe en detalle el proceso de diseño de las arquitecturas Carthago y Carthagonova, desarrolladas para emular en tiempo real el comportamiento de las redes CNN. Se incluyen aspectos relacionados con la estructura y organización de los componentes de su arquitectura externa y los detalles y resultados de la implementación hardware de su arquitectura interna.

Durante el proceso de diseño, se ha puesto especial interés en satisfacer los requisitos de sencillez de las arquitecturas para garantizar la viabilidad de la implementación hardware y asegurar las prestaciones funcionales básicas del sistema. Durante el proceso de diseño, las arquitecturas han sido enfocadas hacia el desarrollo de aplicaciones basadas en redes CNN complejas y de gran tamaño, dedicadas al procesamiento de imágenes de vídeo en tiempo real.

El diseño de las arquitecturas parte del algoritmo utilizado en el capítulo anterior para simular la aproximación de Euler del modelo estándar de la red CNN. En este capítulo, dicho algoritmo es paralelizado y adaptado a las características de una implementación hardware y a su realización sobre dispositivos lógicos reconfigurables. El proceso de adaptación tiene como principal objetivo adecuar el flujo de procesamiento secuencial del algoritmo al esquema de trabajo de los sistemas de vídeo en tiempo real y a las arquitecturas de cómputo paralelas, tanto

desde el punto de vista de la arquitectura externa como de la interna.

Las arquitecturas Carthago y Carthagonova son definidas e implementadas, para unas condiciones de funcionamiento determinadas, mediante la especificación concreta de sus arquitecturas externa e interna, esta última será descrita en detalle a nivel RTL estructural. Los objetivos de esta particularización son verificar la metodología de diseño de las arquitecturas y determinar sus prestaciones en función de las diferentes técnicas de implementación utilizadas.

4.1. Algoritmo de la red CNN

A partir de la aproximación discreta de Euler es posible desarrollar múltiples algoritmos que describan el comportamiento del modelo estándar de la red CNN, uno de los más sencillos es el algoritmo 4.1. Este algoritmo está basado en las ecuaciones 4.1 y 4.2 y ha servido para realizar las simulaciones software de la aproximación de Euler de la red CNN mostradas en el capítulo anterior.

$$Y_{ij}(n+1) = \sum_{k,l \in Nr(ij)} A_{kl} Y_{kl}(n) + \sum_{k,l \in Nr(ij)} B_{kl} U_{kl}(n) + I_{ij} \tag{4.1}$$

$$U_{ij}(n+1) = U_{ij}(n) \tag{4.2}$$

El algoritmo 4.1 ha sido utilizado también como punto de partida para desarrollar las arquitecturas Carthago y Carthagonova, por lo que su naturaleza secuencial ha tenido que ser adaptada a las características del hardware, con las ventajas y desventajas que esto conlleva. Hay que tener en cuenta, además, que este algoritmo no implementa las condiciones de contorno de la red CNN y que, por tanto, tendrá que ser completado con el fin de resolver apropiadamente el comportamiento del modelo. Este aspecto, sin embargo, será tratado con posterioridad en la sección dedicada al desarrollo de la arquitectura interna.

4.2. Consideraciones sobre la adaptación del algoritmo al hardware

El algoritmo 4.1 define con exactitud las tareas de procesamiento que implementarán las arquitecturas. No obstante, dada su naturaleza secuencial, el

Algoritmo 4.1: Aproximación de Euler del modelo de la CNN

```
ENTRADA: Entrada[M,N],Realimentacion[M,N], A[k,l], B[k,l], I, n
SALIDA: Salida[M,N], Entrada[M,N]
r=1;                                  % Radio de vecindad
Salida=ceros[M,N];                    % Salida a cero
Para K=1 hasta n                      % Nueva iteración
  Para i=(1+r) hasta (M-r) hacer        % Barrido de las filas
    Para j=(1+r) hasta (N-r) hacer      % Barrido de las columnas
      Ukl=Entrada[(i-r:i+r),(j-r:j+r)];          % Ventana Ukl
      Ykl=Realimentacion[(i-r:i+r),(j-r:j+r)];   % Ventana Ykl
      Xij=(A⊗Ykl)+(B⊗Ukl)+I;          % Variable de estado
      Si Xij>1 entonces               % Función de activación
        Yij(i,j)=1;                   %            ||
      Si Xij<-1 entonces              %            ||
        Yij(i,j)=-1;                  %            ||
      En caso contrario               %            ||
        Yij(i,j)=Xij;                 %            ||
      FinSi;                          % Fin función de activación
      Salida(i,j)=Yij;                % Actualización de la salida
    FinPara;                          % Fin barrido de las columnas
  FinPara;                            % Fin barrido de las filas
  Realimentacion=Salida               % Realimentación
FinPara;                              % Fin iteración
FIN.
```

algoritmo ha tenido que ser adaptado a las características de una implementación hardware con el fin de sacar provecho al procesamiento paralelo y mejorar las prestaciones de cómputo de la arquitectura. Para ello, y como resultado de nuestra experiencia, a continuación se exponen las principales claves tenidas en cuenta a la hora de realizar la adaptación a los dispositivos FPGA.

- **Aritmética en punto fijo**. El algoritmo debe utilizar aritmética de punto fijo y sustituir el uso de datos en coma flotante. Para realizar este cambio es necesario analizar previamente la precisión aritmética del campo de aplicación en el que se desea trabajar. En determinados casos, como por ejemplo en el ámbito del procesamiento de imágenes, la aritmética entera es suficiente y consigue factores de aceleración entre 10 y 100 veces superior al obtenido por la aritmética en coma flotante.

- **Limitar el tamaño de la memoria**. El conjunto de datos de trabajo debe estar limitado para asegurar su almacenamiento en la memoria interna de la FPGA y minimizar el acceso a los dispositivos de almacenamiento externos.

- **Particionamiento y escalado de matrices**. El algoritmo debe distribuir la información sobre diferentes bloques de memorias independientes que permitan trabajar simultáneamente con varios cauces de datos paralelos y aprovechar el ancho de banda masivo que ofrece la concurrencia del hardware. En la implementación hardware, las variables y matrices usadas por el algoritmo se traducirán en registros y memorias, por lo tanto, si lo que se pretende es utilizar varios datos en paralelo, estos deberán ser colocadas en diferentes variables para poder acceder a ellos de forma simultánea.

- **Perfilado del algoritmo**. Las partes sensibles de ser paralelizadas tienen que ser identificadas y adaptadas para una posible ejecución paralela. Para ello, se deberán aislar las operaciones de cálculo intensivo y minimizar las dependencias de datos entre las tareas de procesamiento.

- **Modo de trabajo en flujo de datos**. Se deberá maximizar el funcionamiento del algoritmo en modo flujo de datos para facilitar la agrupación de tareas en procesos y su segmentación. Esta modificación puede mejorar significativamente la velocidad de cómputo de la arquitectura.

- **Intensificación del uso de los datos**. El algoritmo debe orientarse al uso intensivo de los datos, de manera que el uso de diferentes cauces paralelos permita amortizar el cuello de botella que supone la transferencia de datos desde el exterior de la FPGA. Hay que tener en cuenta, sin embargo, que en algoritmos altamente paralelizables el número de cauces paralelos deberá limitarse en función del compromiso más adecuado entre la réplica de recursos de cómputo y el excesivo consumo de estos.

- **Evitar anidamientos excesivos de los bucles**. Se debe tener en cuenta que, en algunos casos, el anidamiento excesivo de los bucles complicará o hará imposible el funcionamiento y la paralelización del algoritmo en su conjunto. En su lugar, es preferible distribuir el algoritmo en un número de tareas mayor, con cálculos planos y que funcionen en flujo de datos.

4.3. Consideraciones sobre la paralelización del algoritmo

La paralelización del algoritmo consistirá en dividir a este en unidades de cómputo de menor tamaño con el fin de extraer su concurrencia e implementarla en hardware. El proceso de paralelización se ha divido de dos fases: la primera, se encargará de descomponer el algoritmo en unidades de cómputo de pequeño tamaño, que denominaremos tareas, y la segunda, de agrupar las tareas en unidades de cómputo de mayor tamaño, que denominaremos procesos. El objetivo de la agrupación es adaptar la descomposición inicial del algoritmo a las características y necesidades del hardware.

La descomposición del algoritmo se ha realizado en función de los niveles de paralelización definidos anteriormente en la descripción del sistema (sección 3.4.1). En los niveles superiores, incluido el nivel de iteración, la descomposición afectará a la arquitectura externa del hardware, mientras en los niveles inferiores afectará a la arquitectura interna. Para sistematizar el proceso de descomposición se ha recurrido a la utilización de grafos de dependencias, los cuales permiten identificar gráficamente la relación de dependencia entre las tareas.

Una vez que la concurrencia del algoritmo ha sido detectada y representada gráficamente, la fase de agrupación tendrá como objetivo simplificar la descomposición y adecuarla a las características del hardware. La descomposición inicial de un algoritmo suele ofrecer un número de tareas elevado que convendrá reducir, agrupándolas en procesos que simplifiquen el desarrollo de la arquitectura y su implementación. Para agrupar las tareas en procesos se han tenido en cuenta los detalles de la tecnología utilizada para la implementación y la información ofrecida por los grafos de dependencias, que ayudarán a identificar qué tareas pueden ser combinadas en el mismo proceso.

4.3.1. Consideraciones sobre la descomposición en tareas

A la hora de descomponer un algoritmo en tareas es posible utilizar múltiples métodos, algunos de ellos específicos para problemas concretos (Almeida et al., 2008). Dependiendo del método utilizado, la descomposición proporcionará re-

sultados diferentes que en unos casos serán más apropiados para la realización hardware que en otro. Independientemente del método utilizado, un aspecto importante a tener en cuenta son los factores a considerar a la hora simplificar la descomposición del algoritmo. En este sentido, a continuación, se indican los principales factores considerados en la descomposición del algoritmo, la mayoría relacionados con la adaptación de las tareas al hardware y sus conexiones.

- **Descomposición indefinida**. La descomposición será indefinida cuando el número de iteraciones ejecutadas por el algoritmo no venga determinado a priori sino que dependa de cada aplicación. Cuando esto ocurre, parte de las tareas serán creadas durante el desarrollo de la arquitectura y otras durante el diseño de la red CNN. En este caso, aunque el número de tareas no se conocerá inicialmente, sí se sabrá como son dichas tareas y cuál será su estructura de comunicación.
- **Granularidad de la descomposición**. Como regla general, el número de tareas de una descomposición deberá ser lo suficientemente grande como para aprovechar eficientemente los recursos hardware de la tecnología y lo suficientemente pequeño como para mantener la flexibilidad del diseño. Habrá que tener en cuenta que los problemas de la paralelización se incrementarán conforme aumente el número de tareas. Para desarrollar la arquitectura se deberá buscar un número adecuado de tareas, el cual será algo superior al de procesos.
- **Aprovechamiento de los recursos de memoria y de cálculo**. Para llevar a cabo la descomposición del algoritmo se ha dado prioridad el aprovechamiento eficiente de los recursos hardware de la tecnología. Para ello, en unos casos ha sido conveniente que las tareas compartan los mismos recursos hardware, y en otros, que utilicen componentes replicados. Por otro lado, se ha tenido en cuenta que una compartición excesiva puede ocasionar cuellos de botella en las comunicaciones y que la réplica excesiva puede provocar problemas de coherencia en los datos y comprometer la escalabilidad de la arquitectura. Durante la fase de descomposición se ha intentado encontrar el equilibrio más adecuado entre la réplica de recursos y su compartición.
- **Patrón de distribución y conexión de las tareas**. La descomposición del algoritmo puede dar lugar a un patrón de distribución regular, si la

organización de las tareas se mantiene uniforme en el espacio, o irregulares, si dicha organización no guarda ningún tipo de proporcionalidad. Para facilitar el diseño de la arquitectura se ha tratado de conseguir un patrón de conexión regular que minimice las interacciones globales y locales entre tareas. Además, se ha procurado que el flujo de datos entre las tareas circule en un solo sentido para evitar los flujo de datos multidireccionales que complican el control de la arquitectura.

4.3.2. Consideraciones sobre la agrupación de tareas

Las tareas obtenidas tras la descomposición han sido agrupadas en unidades de cómputo de mayor tamaño (procesos) con el fin de simplificar y flexibilizar el diseño de las arquitecturas. La agrupación ha consistido en combinar adecuadamente las tareas relacionadas entre sí, por estructuras de datos o cálculos compartidos, y formar con ellas procesos adaptados a las características del hardware. Una agrupación adecuada permitirá aprovechar eficientemente los recursos de la tecnología y minimizar los costes temporales relacionados con la sincronicación y comunicación entre procesos. Los grafos de dependencias han sido una herramienta de gran utilidad en esta fase, ya que han permitido identificar fácilmente las relaciones de dependencia entre las tareas y simplificar la implementación de la arquitectura. Para llevar a cabo la agrupación se han tenido en cuenta los siguientes aspectos:

- **Reducción del volumen de datos transferidos**. Para simplificar la comunicación entre procesos y facilitar la realización del hardware se ha intentado agrupar en el mismo proceso aquellas tareas que comparten la misma estructura de datos, incluyendo en los procesos los recursos de memoria necesarios para almacenar los datos accedidos con mayor frecuencia.

- **Reducción del número de transacciones**. La transferencia de información entre procesos implica un coste temporal que conviene reducir al máximo. Una forma de conseguirlo es agrupar en el mismo proceso aquellas tareas que comparten datos consecutivos de una misma estructura, y aprovechar cada transferencia para transmitir el conjunto de forma simultánea. Otras dos técnicas empleadas han sido agrupar en el mismo proceso a las tareas que se comunican con mayor frecuencia y utilizar recursos de

almacenamiento intermedios para evitar el acceso sostenido a las memorias compartidas.

- **Coste computacional de los procesos**. En general, las tareas obtenidas tras la descomposición deberán ser agrupadas en procesos con un coste computacional similar. Si el coste difiere mucho entre sí, podrá ocurrir que procesos más rápidos no podrán evolucionar hasta que no lo hagan otros más lentos, lo que repercutirá en una caída generalizada del rendimiento de la arquitectura. Una de las estrategias utilizadas en la agrupación ha consistido en estimar a priori el coste computacional de las tareas.

- **Acceso, tamaño y localización de los datos**. Para la agrupación de tareas se ha tenido en cuenta el tamaño de los datos, su localización y el tipo de acceso (sólo lectura, sólo escritura o lectura y escritura). Esta información es útil para decidir qué tareas pueden ser agrupadas en el mismo proceso con el objetivo de reducir las comunicaciones entre ellos.

- **Réplica de recursos de memoria**. Otro factor tenido en cuenta ha sido la utilización de recursos de memoria adicionales para copiar y distribuir los datos (o parte de estos) sobre diferentes procesos. En el caso de tareas relacionadas entre sí, mediante una estructura de datos compartida, la réplica de recursos de memoria permitirá distribuir los datos sobre diferentes procesos para incrementar el número de cauces y la velocidad de procesamiento. La réplica de recursos de memoria favorece al ancho de banda interno de la arquitectura, aunque compromete su escalabilidad al exigir mayor cantidad de recursos.

- **Réplica y compartición de recursos de cálculo**. Durante el proceso de descomposición pueden surgir tareas cuyo objetivo es realizar operaciones intermedias para otros procesos. En ocasiones será interesante que estas tareas se combinen un único proceso, que serán compartidas por todos los procesos que las demanden, y en otras que dichas tareas se repliquen en múltiples procesos para poder ser ejecutadas en paralelo. La réplica de tareas puede mejorar el rendimiento computacional de la arquitectura aunque también puede afectar a la eficiencia de las comunicaciones, puesto que una mayor cantidad de procesos tendrán que acceder simultáneamente a los mismos datos. Para llevar a cabo una buena agrupación será conveniente

analizar, previamente, si las ventajas de la réplica de tareas compensan los inconvenientes que esta representa para las comunicaciones.

4.4. Paralelización del algoritmo de la red CNN

La arquitectura externa de la red CNN se ha desarrollada a partir de la paralelización del algoritmo 4.1. Para ello, además de los factores mencionados en las secciones anteriores, se han tenido en cuenta los resultados obtenidos por dos métodos de descomposición de carácter general (Almeida et al., 2008): el método de descomposición basado en el dominio de los datos y el método de descomposición basado en el flujo de datos. Cada uno de ellos proporciona una descomposición del algoritmo diferente que ha sido analizada con el fin de determinar su adecuación a la tecnología y las características del sistema.

4.4.1. Paralelización basada en la descomposición del dominio de los datos (arquitectura espacial)

Este método suele utilizarse en la paralelización de algoritmos con estructuras de datos grandes y uniformes, como las utilizadas por las redes CNN. El proceso de descomposición parte de una de las principales estructuras de datos del problema: la de entrada, la de salida o cualquier otra estructura intermedia relevante. A dicha estructura se le aplican las siguientes etapas: dividir la estructura de datos del problema en subdominios (preferiblemente del mismo tamaño), asociar cálculos y datos del algoritmo a tareas, y finalmente, asignar las tareas a cada uno de los subdominios del problema. En general, este método dará buenos resultados cuando sea posible aplicar el mismo procesamiento a todos los subdominios.

Este método ha sido aplicado al algoritmo 4.1 utilizando la estructura de datos de salida, puesto que es la que mejor se adapta a este tipo de descomposición. La máxima paralelización del algoritmo se conseguirá dividiendo dicha estructura en el mayor número de partes iguales, es decir, considerando que cada uno de los elementos de la matriz de salida constituye un único subdominio del problema.

En función de esa división, parte de los datos y de las tareas del algoritmo han sido asignados a cada uno de los elementos Y_{ij} de la matriz de salida. En lo que

respecta a las tareas, cada subdominio ha sido asociado con 2 convoluciones, 2 sumas y una función de activación estándar $f(Xij)$. En cuanto a los datos, cada subdominio tiene asignado una submatriz de elementos de la matriz *Entrada* (Ukl), una submatriz de la matriz de *Realimentación* (Ykl) y la plantilla de la red CNN (A,B,I). A partir de esta descomposición es posible desarrollar el grafo de dependencias que determina la relación entre los datos y las tareas de cada subdominio. La figura 4.1-a muestra un ejemplo para el caso de una celda de radio de vecindad $r = 1$.

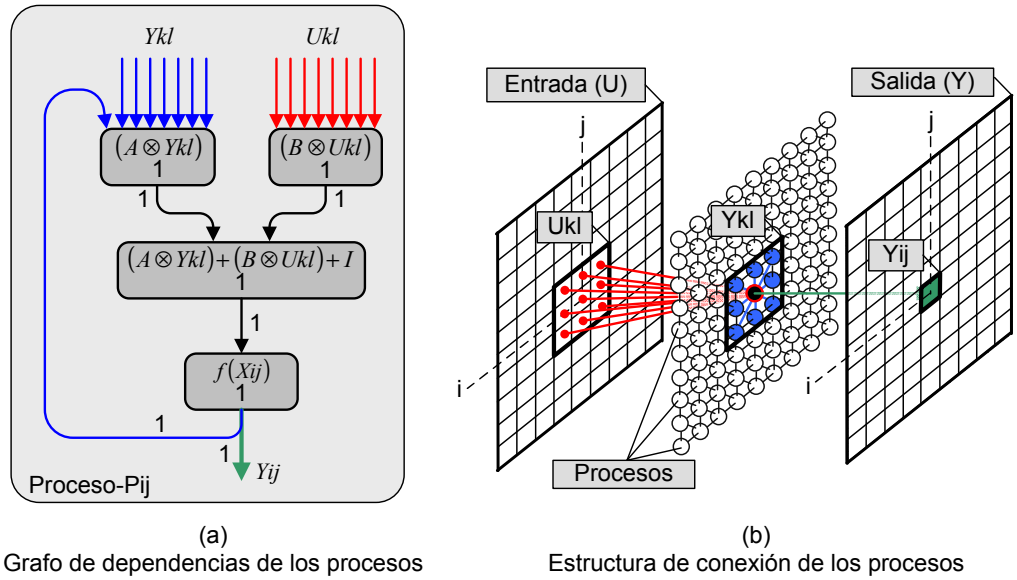


Figura 4.1: Grafo de dependencias y estructura de conexión de los procesos obtenidos en una paralelización basada en el dominio de datos de salida y agrupamiento en celdas individuales.

La agrupación de los datos y las tareas de cada subdominio en un único proceso dará como resultado una arquitectura SIMD de tipo matricial, como la representada en la figura 4.1-b. Esta configuración, adoptada por las CNN-UM, y la mayoría de las realizaciones sobre circuitos ASIC analógicos, maximiza el paralelismo de las redes CNN a nivel de celda y permite obtener las máximas prestaciones en cuanto a velocidad de procesamiento. La idea que subyace bajo este esquema es obtener tantos procesos hardware como celdas tenga la red CNN. Cada proceso estará formado por un circuito realimentado que contribuirá con el resto, en paralelo, en la obtención del resultado. El principal inconveniente de este tipo de arquitecturas es la gran cantidad de elementos de cálculo, memoria,

pines I/O y recursos en general que son necesarios para su implementación, lo que impedirá la realización de redes CNN de gran tamaño sobre FPGAs.

Una alternativa para solucionar este problema consiste en realizar una nueva agrupación y asignar mayor cantidad de celdas a cada proceso. En este caso, cada proceso asumirá las tareas asociadas a una submatriz de celdas y las ejecutará secuencialmente para reducir el consumo de recursos. Este esquema, sin embargo, no conseguirá eliminar todos los inconvenientes asociados con la realización hardware, ya que como se aprecia en la figura 4.2, para el caso de una agrupación de 9 celdas por proceso, tanto la estructura interna de los procesos como su conexión externa resultarán más complejas. En la figura 4.2-b, se aprecia como el cambio a una agrupación de nueve celdas por proceso obliga a pasar de 8 conexiones a 32 conexiones por proceso.

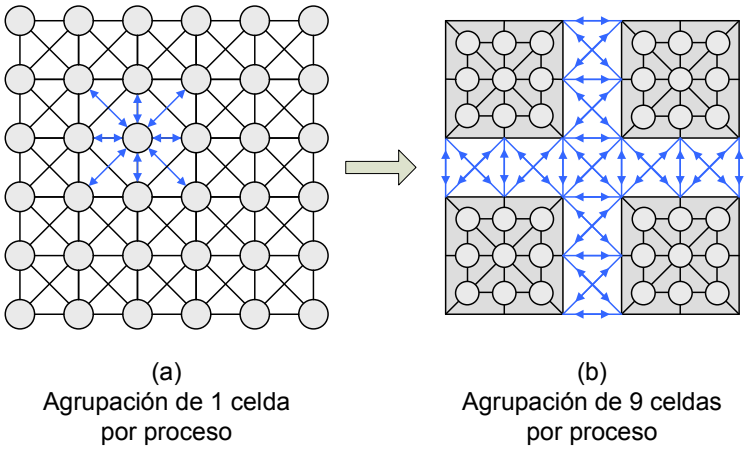


Figura 4.2: Diferentes agrupaciones en una paralelización basada en el dominio de los datos. a) Agrupación de una celda por proceso. b) Agrupación de nueve celdas por proceso.

Cabe concluir, por tanto, que la paralelización basada en la descomposición del dominio de los datos es una solución viable para redes pequeñas y agrupaciones individuales, pero que resulta difícil de implementar en el caso de redes CNN de gran tamaño o cuando las agrupaciones incluyen varias celdas por proceso. Por ello, y a pesar de que este esquema proporciona las máximas prestaciones en cuanto a velocidad de cómputo, este método ha sido descartado para desarrollar de nuestra arquitectura.

4.4.2. Paralelización basadas en el flujo de datos (arquitectura temporal)

Este método se basa en el proceso lógico de ejecución de un algoritmo secuencial que trabaja en flujo de datos. La descomposición del algoritmo por este método será sencilla si el flujo de datos y las dependencias entre las etapas son simples, o compleja, si el algoritmo no presenta una estructura en etapas clara o las comunicaciones entre estas son confusas. La paralelización de un algoritmo compuesto por etapas en cascada se obtendrá directamente, a partir de la propia estructura del algoritmo, una vez identificadas y resueltas las dependencias entre las etapas. Para la descomposición del algoritmo 4.1 se han seguido los siguientes pasos: identificar el flujo de datos principal del algoritmo, identificar las tareas aplicadas al flujo de datos y examinar las dependencias y requisitos de las tareas.

A partir de la observación del algoritmo 4.1 es posible determinar que el principal flujo de datos del algoritmo está formado por las matrices: *Entrada*, *Realimentación* y *Salida*, las cuales tienen que estar completamente actualizadas antes de iniciar una nueva iteración (K). A partir de este flujo datos, y una vez identificadas las tareas del algoritmo, será posible determinar el grafo de dependencias que define al algoritmo. La figura 4.3-a muestra el grafo de dependencia para el caso de una red CNN de radio de vecindad $r = 1$.

Como se observa en dicho grafo, las tareas del algoritmo han sido agrupadas en único proceso recurrente que permitirá emular secuencialmente el procesamiento realizado por la red CNN. Dicho proceso gestiona el flujo de datos y su procesamiento según un esquema de tipo SISD, que puede ser entendido como si una única celda de la red se desplazará de izquierda a derecha y de arriba hacia abajo sobre las matrices de datos que intervienen en el procesamiento (figura 4.3-b).

Este esquema tiene la ventaja de que el consumo de elementos de cálculo se mantiene al mínimo a la vez que se simplifica la interfaz de entrada/salida del sistema. Esto facilitará la conexión con otros componentes y el desarrollo de plataformas multi-FPGAs. No obstante, estas ventajas serán a costa de un mayor consumo de recursos de memoria y una menor velocidad de procesamiento.

Como se observa en la figura 4.3-b, el esquema planteado utilizará tres memorias diferentes (*Entrada*, *Realimentación* y *Salida*) que, en el caso de almacenar

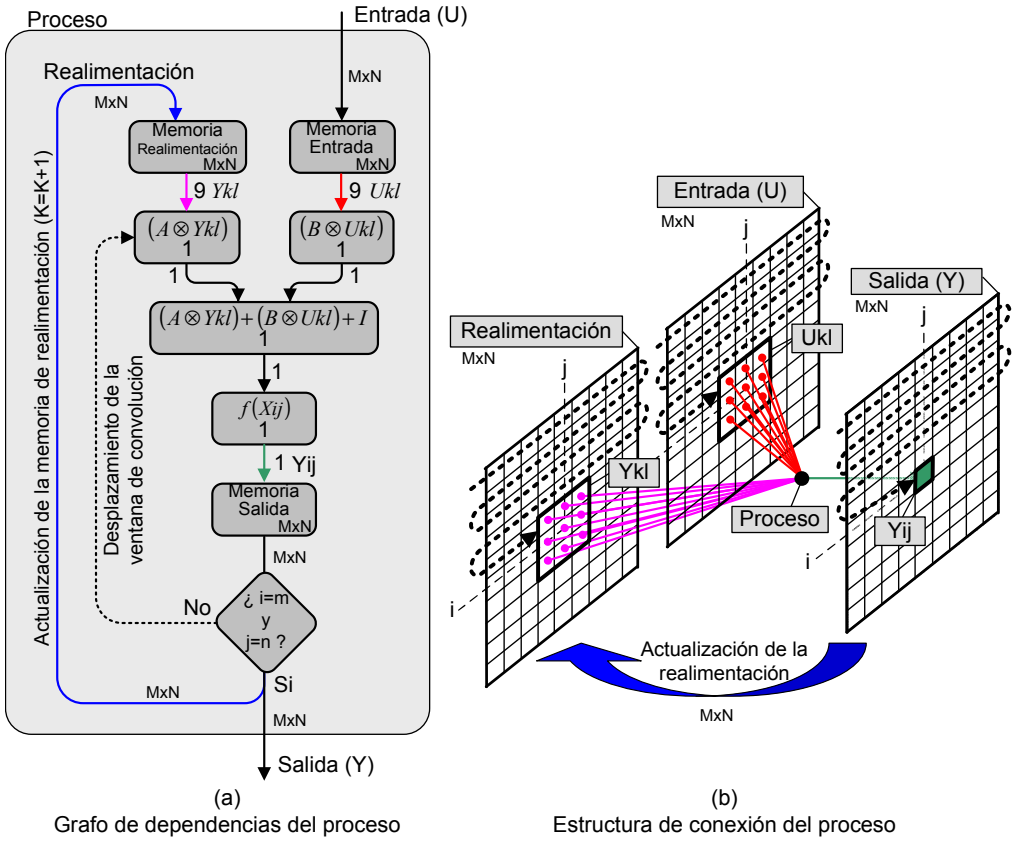


Figura 4.3: Grafo de dependencias y estructura de conexión del proceso obtenido mediante la descomposición basada en el flujo de datos.

imágenes de gran tamaño, hará imposible la implementación del modelo usando exclusivamente las memorias internas de la FPGA. En este caso, será necesario recurrir a dispositivos de almacenamiento externos, lo que complicará la entrada/salida de la arquitectura y la realización hardware. Por otro lado, el bajo número de elementos de cálculo utilizados por el proceso del esquema hará que su velocidad de procesamiento sea muy inferior a la obtenida por las arquitecturas matriciales (figura 4.1) y que, por tanto, su aplicación en sistemas de tiempo real se vea seriamente comprometida.

Una solución para aumentar la velocidad de procesamiento de este tipo de esquemas consiste en replicar las tareas del proceso y construir varios procesos idénticos que serán ejecutados en paralelo. Cada proceso se encargará de procesar una parte de las matrices de datos, de forma similar a como se hace en las arquitecturas Castle y Falcon (secciones 2.7 y 2.9).

En la figura 4.4 se muestran las diferencia entre un esquema formado por un solo proceso (figura 4.4-a) y otro formado por cuatro procesos paralelos (figura 4.4-b). A partir de la figura 4.4-b, es posible deducir que el esquema con cuatro procesos será, aproximadamente, cuatro veces más rápido que el esquema de un sólo proceso. Hay que tener en cuenta, sin embargo, que el patrón de conexión horizontal, que es necesario establecer entre los procesos, exigirá solapar la información entre sus fronteras y supondrá un mayor consumo de recursos de memoria y un sistema de control más complejo.

El esquema con varios flujos paralelos ($F1$,$F2$,$F3$,$F4$) requiere además que las memorias tengan mayor ancho de banda y que sea necesario utilizar unidades de gestión de memoria que eviten los cuellos de botella en el acceso, lo que dificultará el diseño y la implementación de la arquitectura hardware y su aplicación en sistemas de tiempo real.

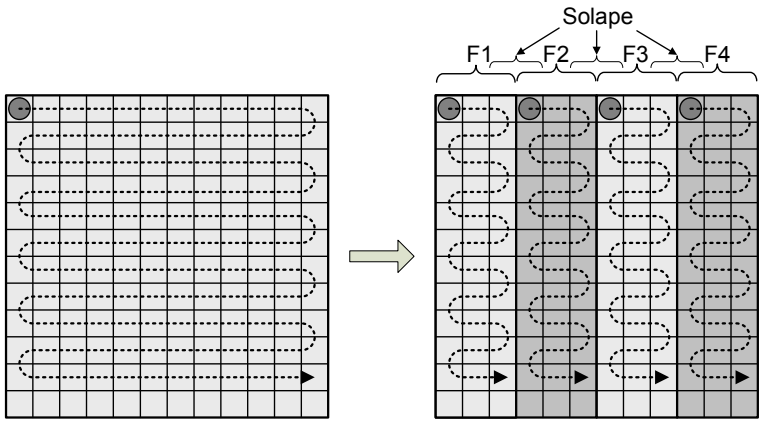


Figura 4.4: Distintas agrupaciones en una paralelización basada en el flujo de datos. (a) Agrupación en un único proceso. (b) Réplica y agrupación en cuatro procesos paralelos idénticos.

4.5. Esquema de procesamiento de la arquitectura Carthago

La arquitectura Carthago ha sido diseñada a partir de un esquema de procesamiento en flujo de datos, formada por un solo proceso, en el que se considera que el modelo en lazo cerrado de la red CNN puede ser aproximado por un número finito de iteraciones (n). Situación que, por otro lado, es habitual en la mayoría de las aplicaciones de procesamiento de vídeo en tiempo real basadas en redes CNN. A partir de este esquema, el algoritmo 4.1 ha sido paralelizado replicando las tareas del bucle (que define el lazo) ejecutando sus n-iteraciones en n-Etapas conectadas en cascada.

El desenrollado del bucle en Etapas dará como resultado una arquitectura sistólica o de frente de onda (según la configuración de la señal de reloj) que permitirá aprovechar las técnicas de segmentación para incrementar el paralelismo y las prestaciones de velocidad de la Etapa. Este esquema, formado por un único flujo de datos, será compatible con el funcionamiento de las cámaras de vídeo progresivas y, por tanto, podrá ser fácilmente adaptado a aplicaciones de procesamiento de vídeo en tiempo real.

Hay que tener en cuenta, no obstante, que el número de Etapas en el que se desenrolle la arquitectura afectará a la precisión de los resultados y al consumo de recursos hardware. Dado que ambos factores se incrementan al aumentar el número de Etapas, durante el diseño de las redes CNN será conveniente encontrar el equilibrio más adecuado entre la ocupación de área de la arquitectura y la precisión en los resultados. La figura 4.5 muestra el esquema de procesamiento propuesto y el desenrollado del proceso recurrente en múltiples Etapas conectadas en cascada.

Este esquema de procesamiento, formado por varias Etapas conectadas cascada, implica un patrón de conexión unidireccional que simplificará el diseño de la arquitectura interna y su realización hardware. Dicho patrón de conexión permitirá que las Etapas puedan trabajar con un conjunto de datos reducido y que, por tanto, se pueda evitar tener que almacenar completamente las matrices que constituyen el flujo de datos. Esta particularidad, permite sustituir las grandes memorias requeridas por el proceso recurrente por pequeños buffers de almacena-

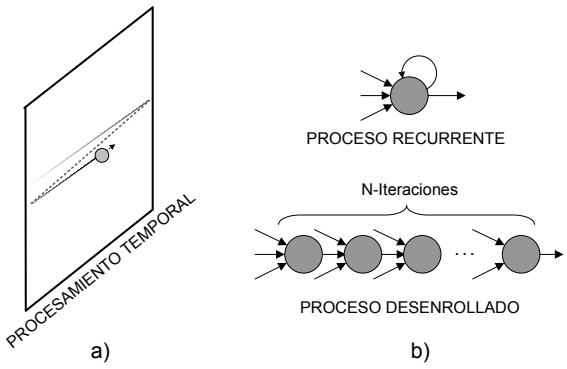


Figura 4.5: Esquema de procesamiento de la arquitectura externa. (a) Procesamiento temporal. (b) Diferentes tipos de procesos: recurrente y desenrollada.

miento local, que serán distribuidos entre todas las Etapas e implementados en el interior de la FPGA. Dichos buffers almacenarán la mínima información requerida por el procesamiento, y su tamaño dependerá tanto del número de columnas de la matriz a procesar como del radio de vecindad de la Etapa.

La figura 4.6 muestra el bloque de información que se almacena en cada uno de los buffer. Su contenido estará formado por los D últimos datos recibidos por la entrada, cuyo número vendrá determinado por la expresión 4.3. Donde r es el radio de vecindad de la Etapa y N el número de columnas de la matriz a procesar.

$$D = (2rN) + (2r + 1) \tag{4.3}$$

El almacenamiento de estos pequeños bloques de datos en el interior de la FPGA no solamente simplificará la interfaz de entrada/salida de la arquitectura sino que también hará que su implementación sea más rápida, dado que el tiempo de acceso de la memoria interna es menor que el de la externa.

Por otro lado, la distribución de Etapas en cascada implica que cada una de ellas podrá utilizar su propia plantilla (A, B e I) y que, por tanto, podrán realizarse procesamiento más complejos basados en la concatenación de múltiples plantillas en cascada. Por otra parte, la sencillez de la interfaz de entrada/salida de las Etapas facilitará la expansión del sistema y permitirá la adaptación de la arquitectura a un gran número de aplicaciones.

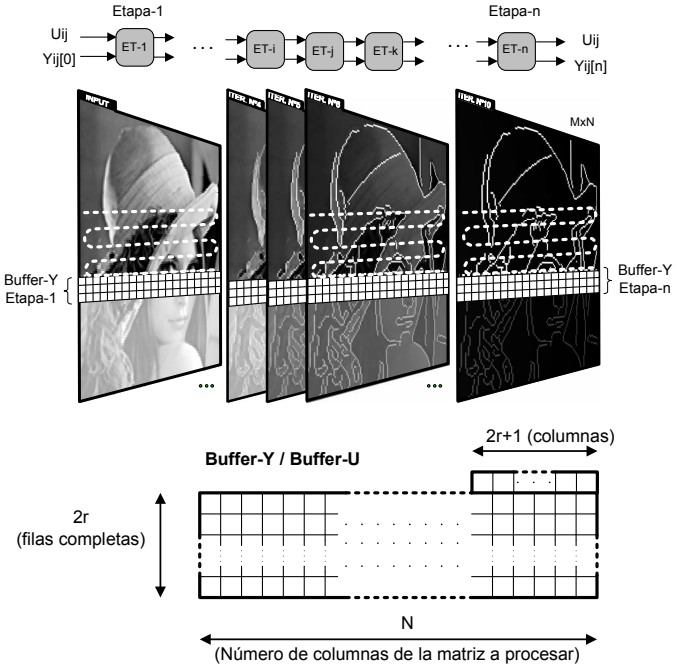


Figura 4.6: Flujo de procesamiento de la arquitectura Carthago y representación del bloque de información almacenado en los Buffers de las Etapas.

4.6. Organización de la arquitectura externa de Carthago

La arquitectura externa de Carthago ha sido ideada para provechar la característica de reconfigurabilidad de los dispositivos FPGAs. El objetivo que se pretende es que dicha arquitectura pueda ser diseñada y modificada de forma rápida y sencilla en función de las características y necesidades de cada aplicación. La arquitectura externa podrá ser especificada por usuario sin conocimientos electrónicos a partir de una estructura flexible y modular basada en la instanciación de cuatro componentes básicos: Etapas, Celdas, Capas y Redes, lo cuales serán organizados jerárquicamente en función de los niveles de abstracción establecidos en el planteamiento del sistema (sección 3.2.1).

La adopción de este esquema de diseño jerárquico permitirá desarrollar estructuras CNN complejas, que podrán ser especificadas a alto nivel mediante herramientas de diseño gráfico como System Generator (Martínez et al., 2005; Martínez et al., 2006), o mediante lenguajes de descripción a nivel de sistema electrónico

(ESL, *Electronic System Level*) como Impulse-C (Martínez et al., 2009).

Como muestra la parte superior de la figura 4.7, el nivel de sistema especificará la conexión de los dispositivos FPGA usados en la aplicación. Conforme nos desplazamos hacia abajo en la jerarquía, se observa que cada FPGA podrá alojar diferentes redes CNN que podrán contener distintas Capas. Dichas Capas estarán formadas por diferentes Celdas que, a su vez, se desenrollarán en Etapas.

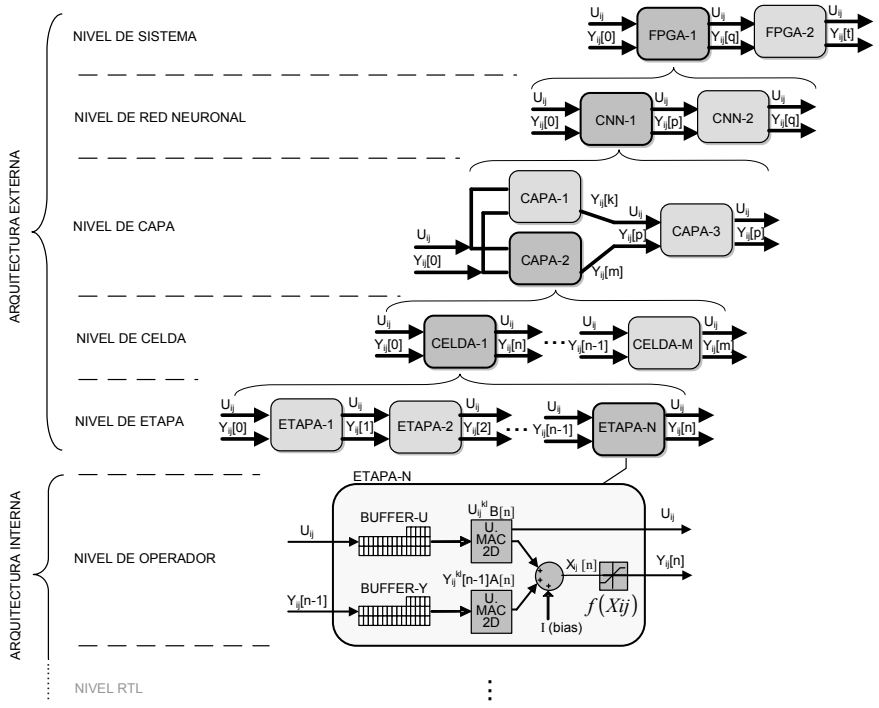


Figura 4.7: Jerarquía de la arquitectura externa propuesta y diagrama de bloques de la Etapa a nivel de operador.

En este esquema jerárquico, la Etapa representa el elemento de más bajo nivel de la arquitectura externa. A partir de esta se construirán los restantes componentes que constituirán las redes CNN. El comportamiento de la Etapa dependerá de las plantillas utilizadas y afectará a la respuesta de los componentes de nivel superior. El paralelismo en la arquitectura externa vendrá determinado por la estructura explícita de la red y por la naturaleza hardware de sus componentes.

En la parte inferior de la figura 4.7, correspondiente al nivel de operador (arquitectura interna), se muestra la descomposición de la Etapa a nivel de operador. Dicha descomposición ha sido representada mediante un diagrama de bloques que

incluye dos buffers locales: Buffer-U y Buffer-Y, que almacenan respectivamente los datos de las entradas Uij e $Yij\,[n-1]$; dos unidades de multiplicación y acumulación bidimensional: MAC-2D-U y MAC-2D-Y; un sumador de tres entradas; y un circuito comparador, encargado de implementar la función de activación estándar $f(X_{ij})$.

La interfaz de entrada/salida de la Etapa está formada por dos puertos de entrada, que se utilizan para introducir los datos a procesar $Yij[n-1]$ e Uij, y dos puertos de salida, que se utilizan para obtener el resultado de la Etapa $Yij[n]$ y el elemento central de la ventana de vecindad de la entrada Uij. La propagación del flujo de entrada Uij por la salida Uij de la Etapa permitirá diseñar redes CNN complejas, compuestas por Etapas en cascada capaces de procesar el mismo flujo de datos de la entrada.

4.6.1. Configuración de las Etapas

La Etapa es el componente de menor nivel de la jerarquía que podrá utilizarse para desarrollar la arquitectura externa de las redes CNN. La versatilidad que ofrece su interfaz de entrada/salida, junto con la posibilidad modificar su plantilla (A, B, I) permitirá configurar Etapas con múltiples modos de operación, los cuales han sido clasificados en tres grupos diferentes (figura 4.8).

Cuando $A = 0$ y $B \neq 0$. En este caso la salida Yij dependerá únicamente de los datos recibidos por la entrada Uij (figura 4.8-a).

Cuando $A \neq 0$ y $B = 0$. En este caso la salida solamente dependerá de los datos recibidos por entrada Yij (figura 4.8-b).

Cuando $A \neq 0$ y $B \neq 0$. En este caso la salida dependerá tanto de los datos recibidos por la entrada Uij como de los recibidos por la entrada Yij (figura 4.8-c).

En general, el caso $A = 0$ y $B = 0$ no tendrá interés práctico debido a que la salida Yij solamente dependerá de la constante I y de la función de activación de la Etapa. En todos los casos, la Etapa propagará el flujo de datos de la entrada Uij al siguiente elemento conectado a la cadena.

La contribución de las entradas Uij e Yij en la salida Yij dependerá, fundamentalmente, de los valores de los operadores B y A, que tendrán que ser

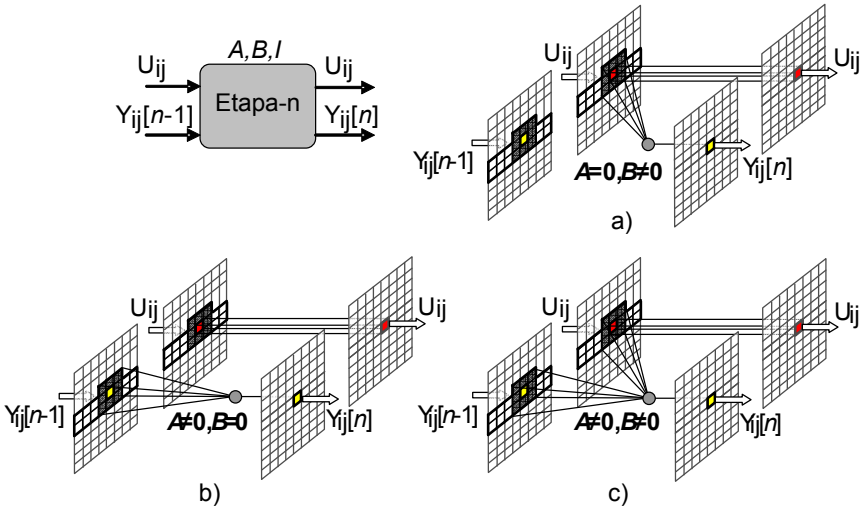


Figura 4.8: Modos de configuración interna de la Etapa. (a) Salida Yij dependiente de la entrada Uij. (b) Salida Yij dependiente de la entrada Yij. (c) Salida Yij dependiente de las entradas Uij e Yij. En todos los casos, la entrada Uij se propagada hacia la salida Uij.

calculados específicamente para cada aplicación. Hay que tener en cuenta que el proceso de diseño de las plantillas será uno de los principales retos a resolver en el desarrollo de aplicaciones basadas en la arquitectura CNN propuesta.

4.6.2. Organización de las Celdas

Las Celdas estarán formadas por una o más Etapas conectadas en cascada. Este tipo de conexión implica que las salidas Uij e Yij de las Etapas anteriores deberán conectarse respectivamente, a las entradas Uij e Yij de las Etapas siguiente, lo que obligará a que todas las Etapas de una misma Celda compartan el mismo flujo de datos Uij. El número de Etapas conectadas en cascada equivaldrá al número de iteraciones de Euler ejecutadas por la Celda, y a su vez, al número de entradas del entorno de vecindad que podrán influir en su salida.

La posibilidad de conectar múltiples Etapas en cascada, cada una de ellas con diferentes plantillas, permitirá desarrollar Celdas complejas que clasificaremos en cuatro tipos diferentes: lineales, no-lineales, variantes en el espacio y variantes en el tiempo, la figura 4.9 muestra un sencillo ejemplo de cada uno de los tipos.

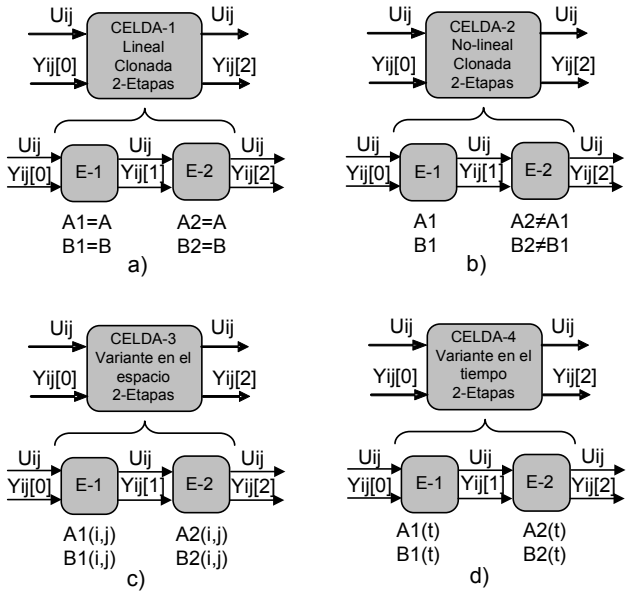


Figura 4.9: Diferentes tipos de Celdas. (a) Celda lineal. (b) Celda no-lineal, (c) Celda variante en el espacio. (d) Celda variante en el tiempo.

Las Celdas lineales se caracterizarán por utilizar Etapas con plantillas idénticas e invariantes tanto en el tiempo como en el espacio (figura 4.9-a). Las Celdas no-lineales también utilizarán plantillas constantes aunque, a diferencia de las anteriores, sus valores serán distintos entre Etapas (figura 4.9-b). Estas dos clases de Celdas se denominarán también Celdas clonadas, debido a que el esquema de procesamiento secuencial empleado aplicará el mismo procesamiento a cada uno de los datos recibidos. Por su parte, las Celdas variantes en el espacio y variantes en el tiempo utilizarán plantillas variables que, respectivamente, modificarán su valor en función de la posición del dato a procesar (i, j) (figura 4.9-c) y del tiempo (figura 4.9-d).

La mezcla de diferentes Etapas en una misma Celda podrá dar lugar a comportamiento dinámicos complejos compuestos por efectos no-lineales y variantes tanto en el espacio como en el tiempo simultáneamente. La figura 4.10 muestra un ejemplo de Celda no-lineal formada por dos Etapas de radio de vecindad 1 (ventanas 3×3). En este caso, la salida Yij de la Celda será función de un número de entradas (5×5) mayor que el definido para la vecindad de una sola Etapa. Según este esquema, la salida Yij de una Celda de n Etapas de radio de vecindad r podrá depender de hasta $(2rn + 1)^2$ entradas diferentes.

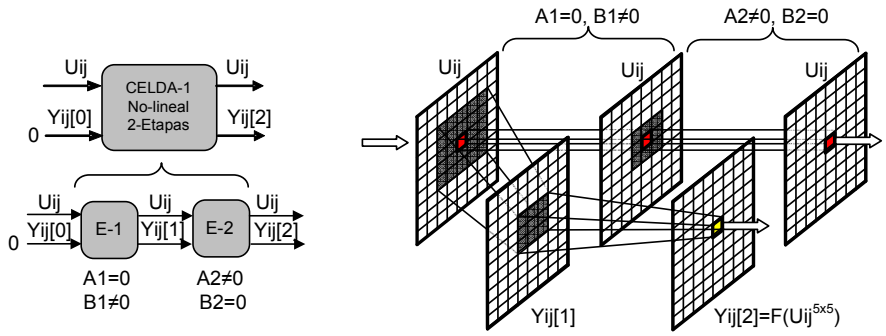


Figura 4.10: Ejemplo de Celda no-lineal con salida Yij dependiente de un número de entradas mayor que el definido por el radio de vecindad de sus Etapas.

4.6.3. Organización de las Capas

Las Capas de las redes CNN estarán constituidas por una o más Celdas que podrán conectarse en cascada o en paralelo, la única condición para que pertenezca a la misma Capa es que todas reciban el mismo flujo de datos por su entrada Uij. La complejidad de las Capas dependerá del número y tipo de Celdas utilizadas (no-lineales, variantes, invariantes, etc.) y de su organización interna, la cual podrá seguir dos esquemas básicos: formado por un solo flujo de datos o formado por varios flujos de datos paralelos, como muestra la figura 4.11.

Las Capas con un solo flujo datos estarán formadas por una o varias Celdas conectadas en cascada, como muestra la figura 4.11-a. En este esquema las Celdas que forman el cauce son identificadas alfabéticamente. La organización interna de este esquema implica que la interfaz de entrada/salida de la Capa tendrá solamente dos puertos de entrada (Uij e Yij) y dos puertos de salida (Uij e Yij). En las Capas con varios flujos de datos (figura 4.11-b) el número de puertos de la interfaz dependerá del número de cauces implementados. En la figura 4.11-b se observa que las Celdas del mismo cauce comparten el mismo índice numérico.

En el caso de que una Capa esté formada por varios flujos paralelos, como sucede en el ejemplo de la figura 4.11-b, será necesario considerar el retraso en cada cauce para garantizar la convergencia de la información en la salida. Dicho retraso vendrá determinado por el número de Etapas de cada cauce y podrá ser igualado añadiendo Etapas adicionales a los cauces con menor retraso. El objetivo de estas Etapas no será realizar procesamiento sino retrasar la información.

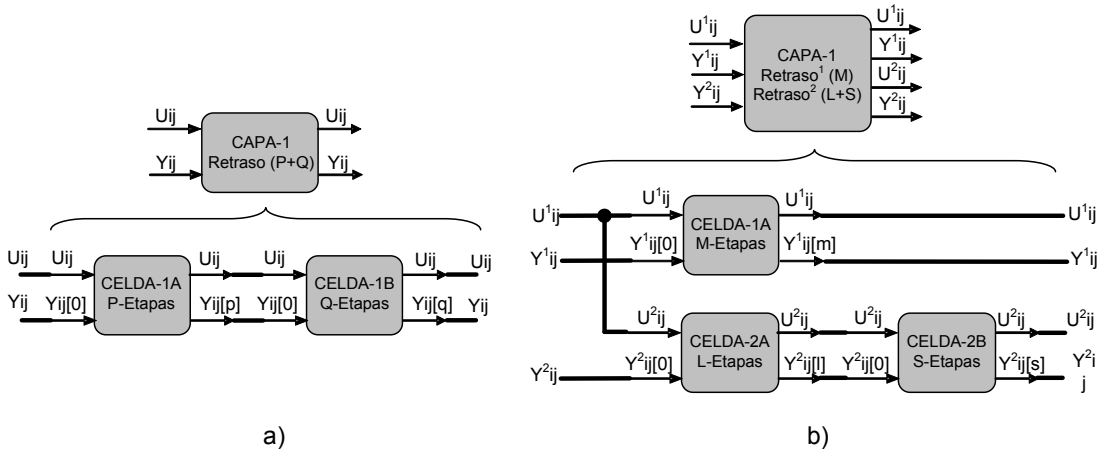


Figura 4.11: Ejemplos de la organización interna de dos tipos de Capas diferentes. a) Capa con uno solo flujo de datos. b) Capa con dos flujos de datos paralelos.

4.6.4. Organización de las Redes

Las redes CNN basadas en la arquitectura Carthago podrán ser de dos tipos: mono-capa o multi-capa, la figura 4.12 muestra un sencillo ejemplo para cada uno de los casos. Las redes mono-capa (figura 4.12-a) tendrán una sola Capa que podrá contener una o más Celdas conectadas en paralelo o en cascada. Las redes multi-capa (figura 4.12-b) estarán formadas por varias Capas cuyas fronteras estarán delimitadas por la conexión de la salida Yij de una Celda con la entrada Uij de la siguiente Celda. La sucesión de Celdas según este esquema definirá el número Capas de la red CNN y su organización interna.

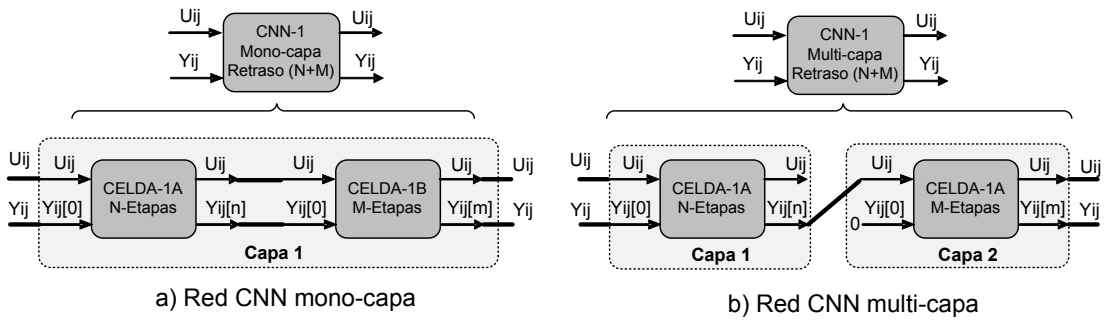


Figura 4.12: diferentes tipos de redes CNN: a) Red mono-capa formada por dos Celdas. b) Red multi-capa formada por Capas de una Celda.

La modularidad y versatilidad que ofrece la arquitectura externa, permitirá que cada Red CNN pueda presentar su propia estructura en función de las particularidades y necesidades de cada aplicación. Esta característica, junto con la posibilidad de modificar las plantillas de las Etapas, hará posible el diseño de redes CNN de gran tamaño y complejidad de forma rápida y sencilla.

La figura 4.13 muestra un sencillo ejemplo de Red CNN de tres capas, representa en tres niveles de abstracción diferentes: a nivel de red, a nivel de capa y a nivel de celda. Como se aprecia en el nivel inferior, la primera Capa dispone de tres Celdas repartidas en dos cauces de procesamiento paralelos. Las otras dos Capas tienen una única Celda y un solo flujo de datos. Para garantizar que las salidas de las Capas 1 y 2 convergen simultáneamente en la entrada de la Capa-3, la suma de los retrasos de la Celda-1A y Celda-1B de la Capa-1 debe ser igual a la suma de los retrasos de la Celda-2A y Celda-1A de las Capa-1 y 2 respectivamente, es decir, $(N + K) = (M + L)$. Donde N, K, M representan el retraso de las Celdas 1A, 1B y 2A de la Capa-1 y L el retraso de la Celda-1A de la Capa-2.

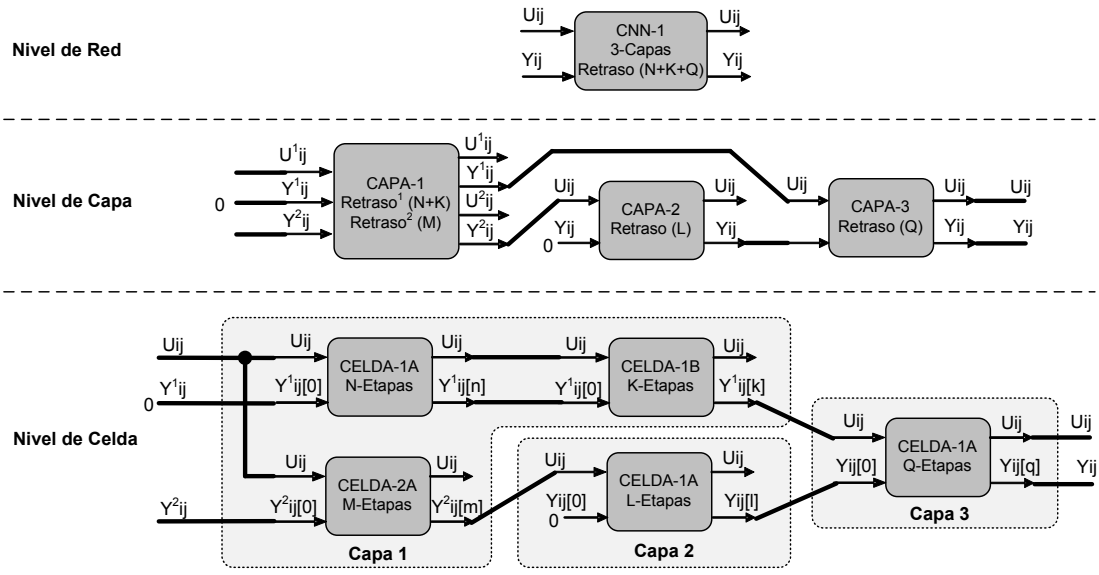


Figura 4.13: Ejemplo de red CNN de 3 capas representada en tres niveles de abstracción distintos: nivel de red, nivel de capa y nivel de celda.

4.7. Etapa Carthago

La Etapa es el componente de menor nivel de la arquitectura externa y surge de la paralelización del algoritmo 4.1. Dado que durante el proceso de paralelización la estructura de datos del algoritmo ha sido modificada, pasando de estar formada por tres grandes matrices (*Entrada*, *Realimentación* y *Salida*) a estar distribuida sobre un número indefinido de pequeños buffers locales, ha sido conveniente plantear un nuevo algoritmo que describa, de forma más apropiada, la arquitectura interna de la Etapa.

A pesar de que las redes CNN pueden ser desarrolladas combinando distintos tipos de Etapas (con diferente arquitectura interna e interfaces de entrada/salida), en aras de una mayor simplicidad y uniformidad se ha optado por que todas utilicen la misma arquitectura interna. Por otro lado, dado que la Etapa es el componente que soporta la carga del procesamiento, habrá que tener en cuenta que un diseño eficiente mejorará considerablemente las prestaciones del sistema, es decir, reducirá el consumo de recursos hardware e incrementará la velocidad de procesamiento.

La funcionalidad de la arquitectura interna de la Etapa puede ser definida a partir de multitud de algoritmos distintos. No obstante, para simplificar el diseño y la implementación se ha optado por desarrollar un algoritmo sencillo fácilmente paralelizable y adaptable a la tecnología. Para llevar a cabo la descomposición del algoritmo en tareas y la agrupación en procesos se han tenido en cuenta los factores enunciados en las secciones 4.3.1 y 4.3.2.

4.7.1. Algoritmo de la Etapa Carthago

El algoritmo de la Etapa Carthago está basado en el funcionamiento de los buffers circulares utilizados para almacenar los datos recibido por las entradas Uij e Yij. Cada buffer está compuesto por dos memorias RAM: una de doble puerto, denominada RAM1, y otra de simple puerto, denominada RAM2. La RAM1 almacena $2rN$ datos, siendo N el número de columnas de la matriz de entrada y r el radio de vecindad. Mientras que la RAM2, de menor tamaño, almacena los restantes $2r + 1$ datos requeridos por la Etapa (ver figura 4.6).

Los buffers circulares de las dos entradas funcionan exactamente igual y presentan una estructura de almacenamiento en anillo, es decir, conforme los datos son recibidos son almacenados en posiciones de memoria consecutivas, cuando se llega al límite de la memoria, los nuevos datos son almacenan desde el principio sobrescribiendo a los anteriores.

Este modo de almacenamiento se consigue empleando tres contadores circulares: uno ascendente, denominado Contador-a, y dos descendentes, denominados Contador-b y Contador-d. El Contador-a y el Contador-b tienen módulo igual al número de columnas de la matriz a procesar (N) y son utilizados, respectivamente, como punteros de escritura y lectura de la memoria RAM1. El Contador-d tiene módulo $2r + 1$ y se utiliza tanto de puntero de escritura como de puntero de lectura de la memoria RAM2. Parte de la información leída de los buffers será nuevamente realimentada a las memorias a través de un banco de $2r$ registros, denominado BanReg-RLU o BanReg-RLY según la entrada. La figura 4.14 ejemplifica el funcionamiento interno de los Buffer circulares para el caso de una Etapa de radio de vecindad 1 y matrices de entrada de tamaño $M \times N$.

La figura 4.14-a muestra el instante en el que el dato U_{66} es almacenado en la fila inferior de la RAM1, concretamente en la columna seleccionada por el Contador-a ($a = 6$). Simultáneamente, en la fila superior de dicha memoria y en la RAM2, se almacenan los datos guardados en el banco BanReg-RLU, el cual contiene a los vecinos superiores del dato U_{66} (U_{56} y U_{46}), almacenados en este registro en la iteración anterior.

Finalizada la escritura, a continuación se llevarán a cabo $2r + 2$ lecturas consecutivas sobre cada buffer. La lectura sobre la RAM1 tendrá como puntero de direcciones el Contador-b, inicializado con el valor del Contador-a tras ser incrementado en la escritura ($b = a$), mientras que la lectura de la RAM2 usará como puntero el Contador-d. El Contador-b se decrementará $2r + 1$ veces, para leer un total de $2r+2$ columnas de la RAM1, mientras que el Contador-d se decrementará $2r$ veces, para leer $2r + 1$ columnas de la RAM2.

Las figuras 4.14-b,c,d y e muestran el proceso de lectura de cuatro columnas consecutivas del Buffer-U. Concretamente, se leen la columna $b = 7$ (figura 4.14-b), que contiene los vecinos superiores del dato siguiente (U_{57} y U_{47}) y las 3 columnas anteriores $b = 6$ y $d = 2$; $b = 5$ y $d = 1$; $b = 4$ y $d = 0$, que contienen los

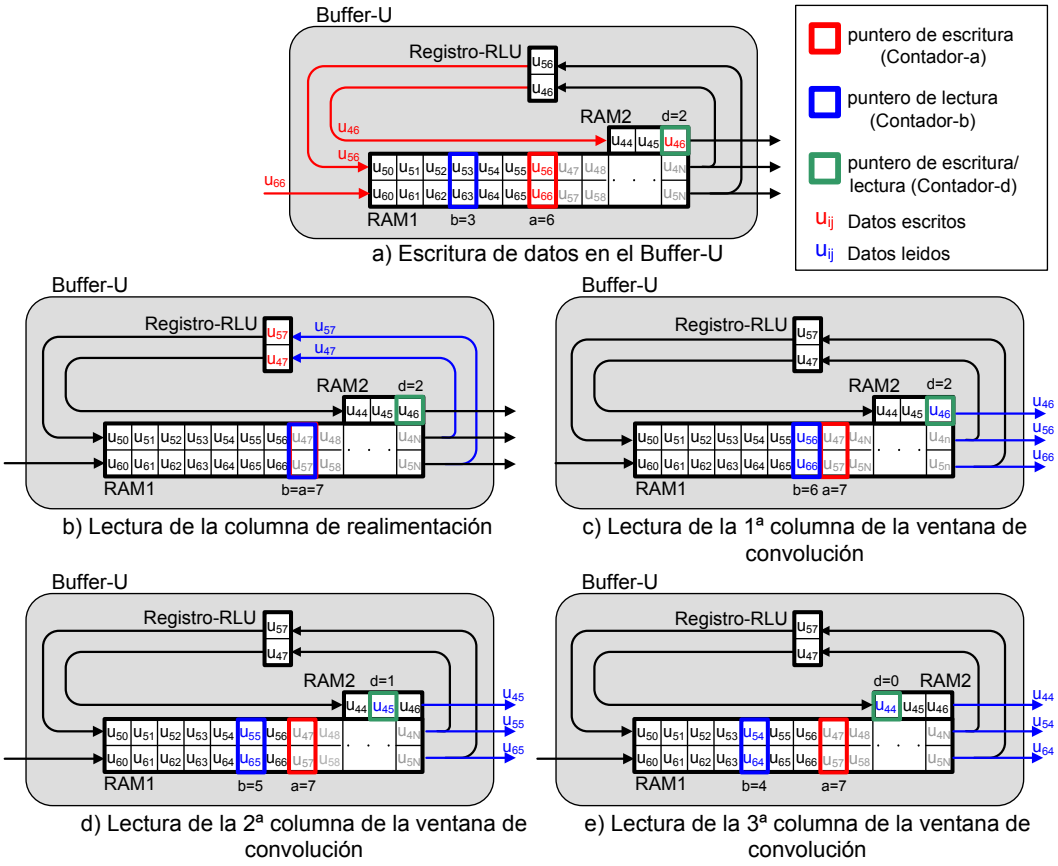


Figura 4.14: Secuencia de escritura y lecturas sobre el Buffer-U de la Etapa Carthago (Etapa de radio de vecindad $r = 1$ y matrices de tamaño $M \times N$).

datos de la ventana de vecindad actual (figura 4.14-c, d y e). En Dicha figura se puede deducir también que la latencia introducida por el buffer será de $(Nr + 2r)$ datos, dado que para el dato de entrada U_{66} la ventana de vecindad leída estará centrada en el dato U_{55}.

El algoritmo completo de la Etapa se muestra en la figura 4.15, su ejecución se iniciará en el instante en el que se reciban simultáneamente los datos Uij e Yij y se almacenen en los Buffer-U y Buffer-Y respectivamente. A continuación, tras incrementar el Contador-a y actualizar en Contador-b, la primera columna extraída de cada buffer (únicamente de la RAM1) será almacenada, respectivamente, en los bancos BanReg-RLU y BanReg-RLY, para posteriormente, en el siguiente ciclo de escritura, realimentarla a la RAM1. Una vez decrementado el Contador-b, y puestos a cero tanto el Contador-c como las variables de la mul-

tiplicación (MAC-U y MAC-Y), se ejecutará el bucle encargado de extraer las $2r+1$ columnas de la ventana de vecindad. El número de iteraciones del bucle estará controlado por el Contador-c que se iniciará a 0 y se incrementará hasta $2r$. Dentro del bucle se llevarán a cabo 6 pasos consecutivos:

El primer paso consistirá en leer de cada buffer la columna seleccionada por los contadores de dirección: Contador-b (RAM1) y Contador-d (RAM2). La columna será almacenada a continuación en un banco de $2r+1$ registros: el banco BanReg-BU para el Buffer-U y el banco BanReg-BY para el Buffer-Y. Posteriormente, el contenido de estos bancos será multiplicado con los operadores B y A, respectivamente, y el resultado almacenado en las matrices MatReg-OPB y MatReg-OPA, ambas de dimensión $(2r+1)^2$.

El segundo paso del bucle consiste en comprobar si la columna extraída del Buffer-U coincide con la columna central de la ventana de vecindad (Contador-c=r). En ese caso, el elemento central de dicha columna será almacenado en el registro Reg-U para posteriormente propagarlo hacia la salida U de la Etapa.

En los dos pasos siguientes del bucle se comprobará si la columna extraída del buffer pertenece al interior de la matriz de entrada, o si por el contrario, algunos de sus elementos pertenecen a la frontera. Para llevar a cabo esta comprobación se utilizarán dos contadores: el Contador-i y el Contador-j, los cuales determinan, respectivamente, a que fila y columna de la matriz de entrada pertenece el último dato recibido por la Etapa. Si el Contador-j determina que la columna es frontera lateral, el contenido de los bancos BanReg-BU y BanReg-BY será puesto a cero, para evitar que intervengan en el procesamiento. Si el Contador-i indica que hay elementos frontera superiores o inferiores, entonces, las filas correspondientes de los bancos de registros BanReg-BU y BanReg-BY serán puestas a cero. Estas comprobaciones servirán para establecer las condiciones de contorno Dirichlet de la red CNN.

El quinto paso se encargará de llevar a cabo las operaciones de multiplicación y acumulación (MAC) de la Etapa. Por una lado, el contenido del banco BanReg-BU será multiplicado escalarmente con el contenido de la columna de la matriz MatReg-OPB seleccionada por el contador-c. Y por otro, esa misma operación es realizada entre el banco BanReg-BY y la misma columna de la matriz MatReg-OPA. Los resultados de estas operaciones serán acumulados en las variables MAC-

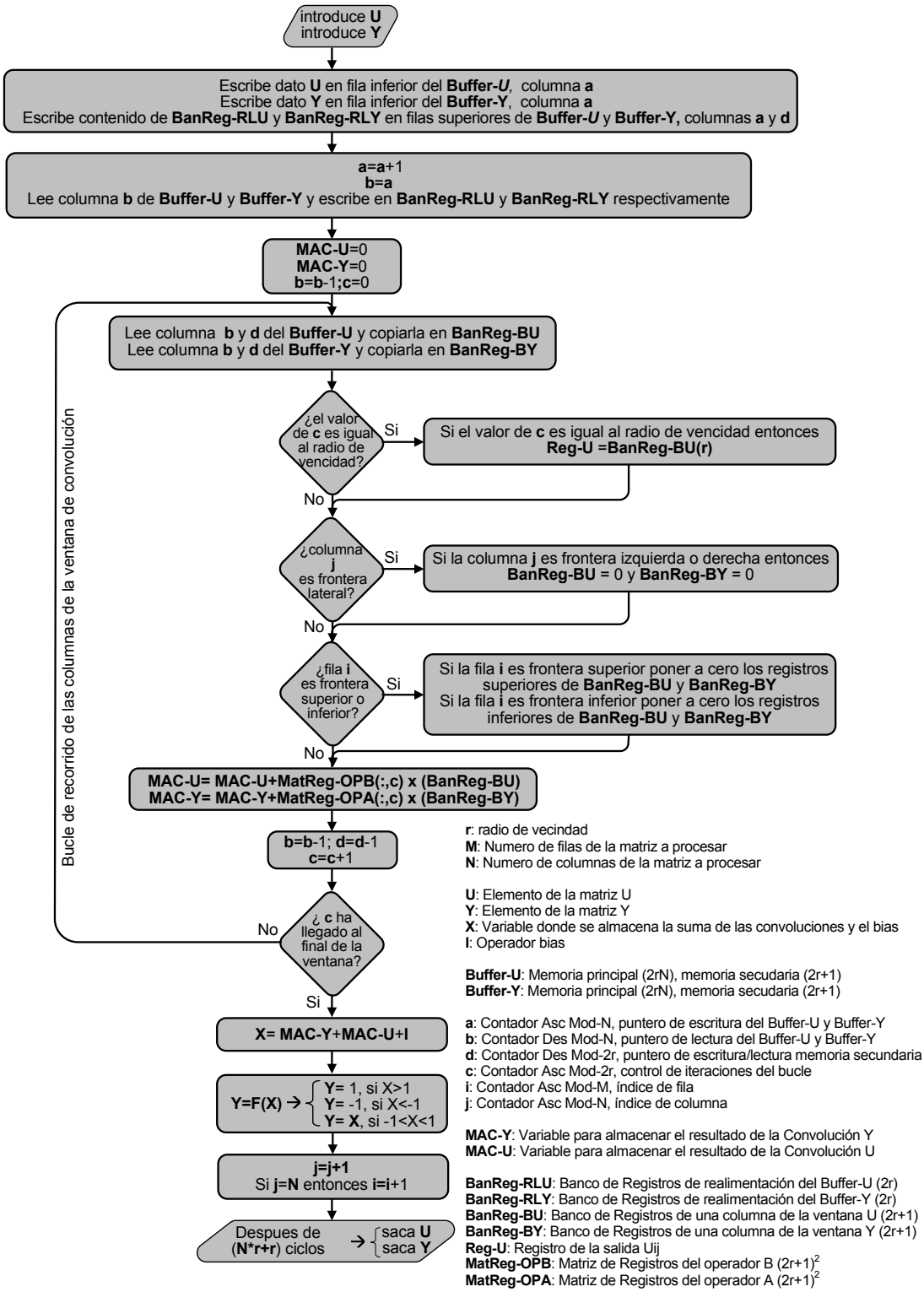


Figura 4.15: Algoritmo de la Etapa Carthago

U y MAC-Y respectivamente, donde una vez finalizado el bucle quedarán los resultados de las operaciones MAC.

El último paso del bucle consistirá en decrementar los punteros de lectura de los buffers (Contador-b y Contador-d) e incrementar el contador de interacciones del bucle (Contador-c), el cual será evaluado a continuación para determinar si se tiene que repetir el bucle o no.

Una vez finalizado el bucle, los resultados almacenados en las variables MAC-Y y MAC-U son sumados al operador de bias I para obtener el valor de la variable de estado de la Etapa X. Dicha variable servirá como argumento de entrada de la función de activación estándar $f(X)$, la cual a su vez, devolverá el valor de salida Y de la Etapa. El algoritmo concluirá con el incremento del contador de columnas Contador-j, y en el caso de que éste desborde, también con el del contador de filas Contador-i. Debido a la latencia de los buffers, las salidas de la Etapa (Uij e Yij) no deberán ser habilitadas hasta que no se hayan procesados los $Nr + r$ primeros datos, siendo N el número de columnas de la matriz a procesar y r el radio de vecindad.

4.7.2. Paralelización de la Etapa Carthago

La paralización de la Etapa Carthago ha sido obtenida tras la descomposición del algoritmo 4.15 en tareas y su posterior agrupación en procesos. La figura 4.16 muestra el grafo de dependencias y la agrupación en procesos obtenida.

Como se muestra en la figura 4.16, la descomposición del algoritmo proporciona un conjunto de tareas con dependencias tanto de datos como de control. Las tareas han sido agrupada en cuatro procesos diferentes: el Proceso-U, el Proceso-Y, el Proceso-Salida y el Proceso-Control. Los tres primeros tienen como objetivo almacenar y procesar la información, por lo que son recorridos por el flujo de datos. Mientras que el cuarto se encarga de controlar la secuencia de ejecución de los anteriores. El Proceso-U y el Proceso-Y son similares y únicamente se diferencian en la tarea T15, la cual tiene como misión generar la salida Uij de la Etapa. El Proceso-Salida recibe simultáneamente los resultados del Proceso-U y del Proceso-Y y los opera para generar la salida Yij de la Etapa.

La descomposición del algoritmo se ha centrado en conseguir un patrón de co-

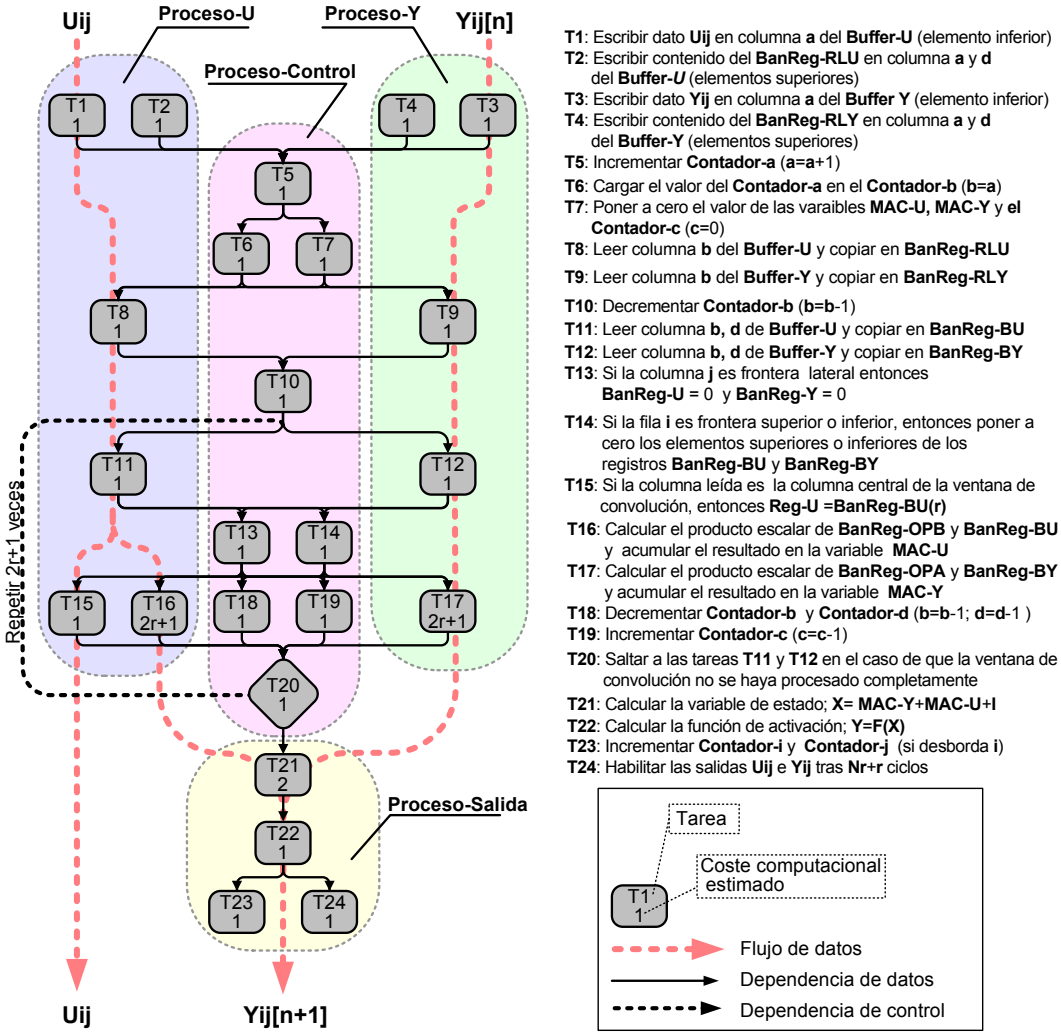


Figura 4.16: Paralelización de la Etapa Carthago: descomposición en tareas y agrupación en procesos.

nexión regular y unidireccional, que simplifique las comunicaciones y que facilite el encauzamiento de los datos que fluyen a través de las tareas. La granularidad de la descomposición se ha adaptado al nivel de operador de la Etapa, buscando optimizar el uso de los recursos hardware y mejorar del compromiso entre la réplica y el uso compartido de recursos. La agrupación de tareas ha dado lugar a dos procesos concurrentes (Proceso-U y Proceso-Y) con un coste computacional equilibrado y fácilmente adaptable a las características del hardware. Los principales criterios considerados en la agrupación de tareas han sido simplificar las comunicaciones entre los procesos, minimizar el volumen de datos transferidos,

y conseguir una secuencia lógica en el almacenamiento y el procesamiento de los datos. La compartición de recursos ha jugado un papel importante en la creación del Proceso-Control dado que en él se han combinado todas las tareas de control y sincronización del Proceso-U y del Proceso-Y.

4.8. Implementación de la Etapa Carthago

En las siguientes secciones se introducen diferentes implementaciones hardware de la Etapa Carthago, particularizadas para las condiciones propuestas en el planteamiento del sistema: Etapas de radio de vecindad $r = 1$ e imágenes de entrada de hasta 1024×1024 píxeles codificadas con 256 niveles de gris. Las implementaciones se han realizado utilizando diferentes técnicas y herramientas de desarrollo, por lo que cada una de ellas ofrece características y prestaciones diferentes. Los objetivos que se persiguen son comprobar el grado de implementabilidad y adaptabilidad de la arquitectura y determinar su eficiencia en cuanto a consumo de recursos hardware y velocidad de procesamiento.

4.8.1. Descripción de la Etapa Carthago a nivel RTL

La arquitectura interna de la Etapa Carthago ha sido diseñada para utilizar estructuras de multiplicación y acumulación recursivas (MAC). Esta solución permite minimizar el consumo de área del circuito a consta de penalizar su velocidad de procesamiento, ya que la unidades MAC requieren múltiples iteraciones para completar las operaciones. La figura 4.17 muestra la estructura completa de la Etapa Carthago a nivel RTL.

Para evitar que el procesamiento secuencial de la arquitectura sea un impedimento para desarrollar aplicaciones en tiempo real se ha optado por aprovechar las ventajas de las unidades super-escalar y super-segmentadas para diseñar la unidad de proceso de la Etapa. Para ello, la arquitectura interna ha sido divida en tres sub-etapas (S1, S2 y S3). La sub-etapa S1 realiza la carga de datos de la Etapa, la sub-etapa S2 realiza las operaciones de procesamiento en dos cauces paralelos super-segmentado y la sub-etapa S3 combina los resultados de la sub-etapas anteriores y genera las salidas de la Etapa.

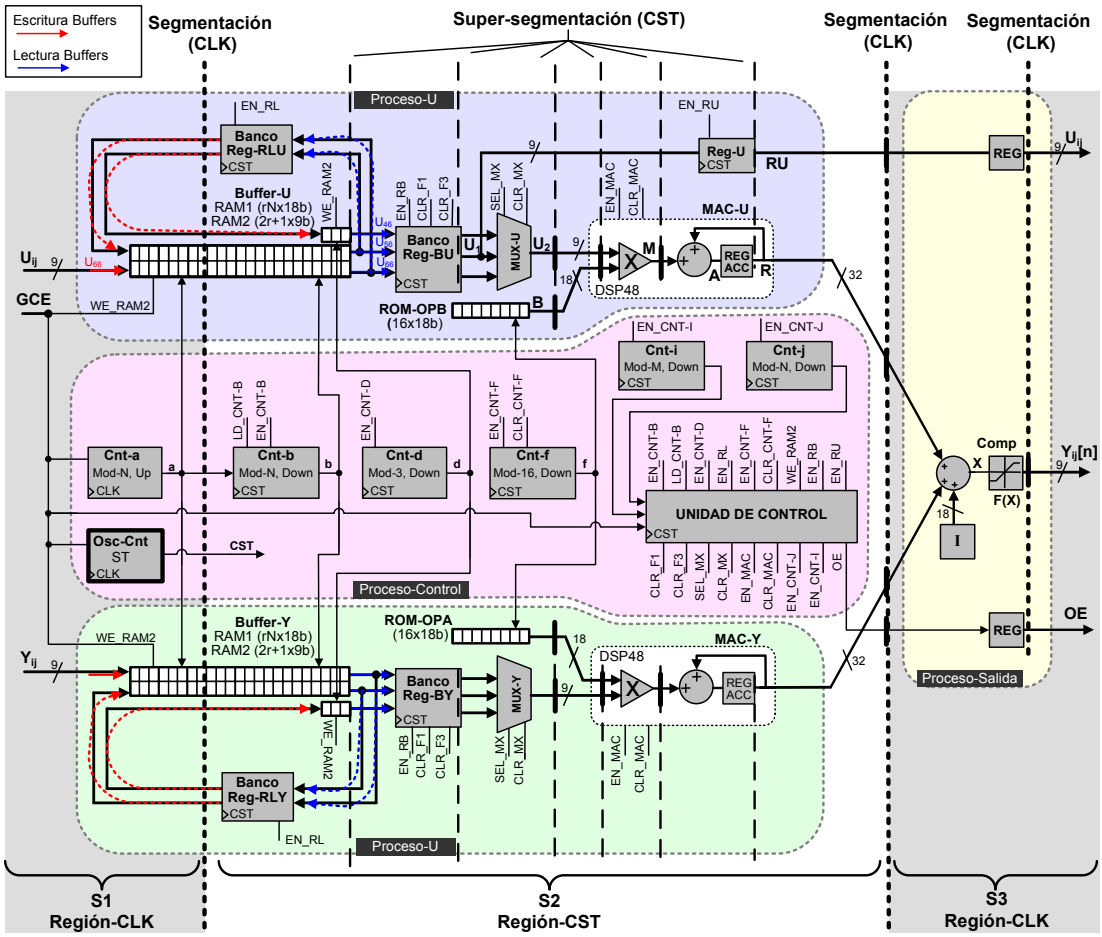


Figura 4.17: Descripción RTL de la Etapa Carthago para un radio de vecindad $r = 1$.

Para sincronizar el circuito se han definido dos regiones de reloj. La región del reloj principal, que incluye las sub-etapas S1 y S3, y la región del reloj secundaria, que incluye solamente la sub-etapa S2. La región de reloj principal utiliza una señal de reloj externa (CLK) para sincronizar el muestreo de los datos de entrada y salida de la Etapa. Y la región de reloj secundaria utiliza una señal de reloj interna (CST), de mayor frecuencia, para sincronizar el circuito super-segmentado encargado de procesar los datos. La utilización de estas dos señales de reloj permite que la sub-etapa S2 pueda procesar los datos en el mismo periodo de reloj en el que los recibe, es decir, con cero latencia con respecto al reloj principal CLK.

Como muestra la figura 4.17, tanto el Proceso-U como el Proceso-Y utilizan tres tipos de memorias para almacenar los datos requeridos por el procesamiento.

Las memorias RAM1 y RAM2 son utilizadas por los buffers circular para almacenar las 3 últimas filas de datos de las matrices de entrada recibidas por la Etapa (la U o la Y según el proceso). La memoria ROM, por otra parte, es utilizada para almacenar los operadores de la plantilla (el OPA o el OPB según el proceso).

La RAM1 ha sido definida como una memoria RAM de doble puerto de 1024×18 bits y es usada para almacenar las 2 últimas filas de datos recibidas de la matriz de entrada. En los 9 bits menos significativos se almacenan los datos de la última fila, mientras que en los 9 bits más significativos se almacenan los datos de la penúltima fila. La escritura de datos en esta memoria se realiza a través del Puerto-A (sincronizado con el reloj CLK) y la lectura a través del Puerto-B (sincronizado con el reloj CST). La RAM2 ha sido definida como una memoria RAM de simple puerto de tamaño 3×9 bits. Esta memoria está sincronizada con el reloj CST y almacena los tres últimos elementos recibidos de la antepenúltima fila de la matriz de entrada. La memoria ROM, también de simple puerto y síncronizada con el reloj CST, tiene una capacidad de 16×18 bits.

El acceso a las memorias se realiza mediante diferentes contadores conectados a sus correspondientes buses de dirección, incluidos en Proceso-control. Dado que tanto el Proceso-U como el Proceso-Y trabajan simultáneamente, los contadores son compartidos por ambos procesos para reducir el consumo de recursos hardware. Para el direccionamiento de las memorias de doble puerto se utilizan dos contadores de modulo-N: el Contador-a (ascendente), conectado al Puerto-A, y el Contador-b (descendente), conectado al Puerto-B. El direccionamiento de las memorias RAM2 y ROM de simple puerto se lleva a cabo mediante dos contadores descendentes: el Contador-d de 2 bits, usado por la RAM2, y el Contador-f de 4 bits, usado por la memoria ROM. Finalmente, otros dos contadores descendentes de 10 bits son utilizados para determinar la posición de los datos recibidos por la Etapa dentro de la matriz de entrada, el Contador-i indica la fila y el Contador-j la columna.

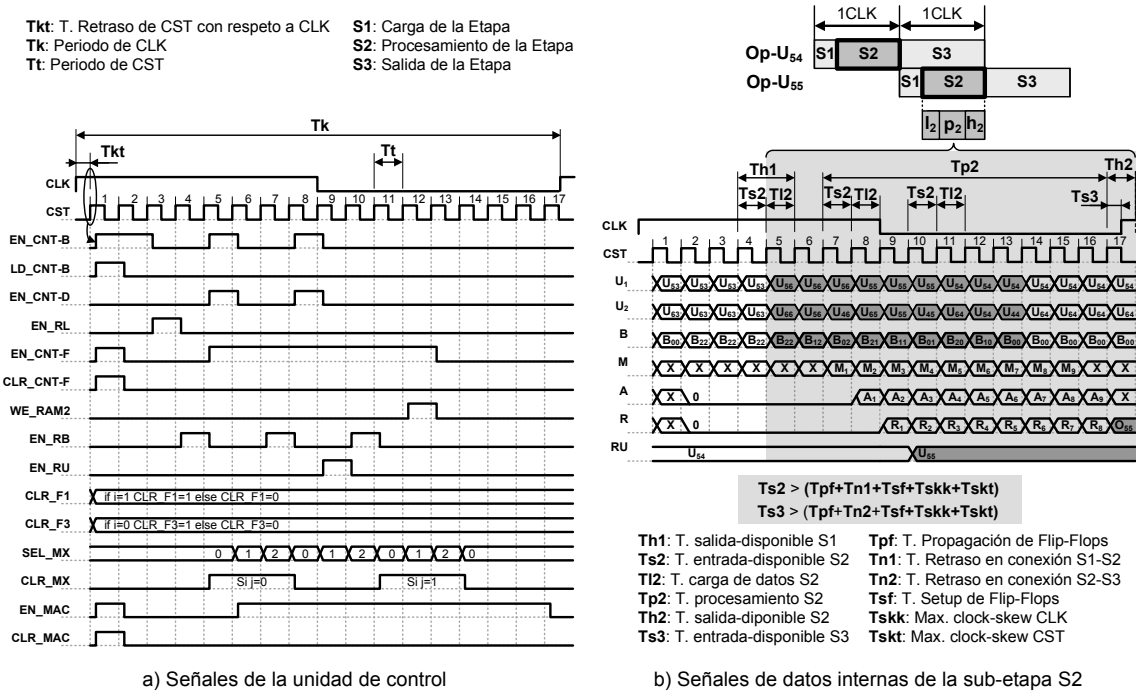
El Proceso-U y el Proceso-Y incluyen además dos bancos de registros. El primero, de tamaño 2×9 bits (BanReg-RLU y BanReg-RLY respectivamente), está conectado a la entrada y a la salida del buffer circular. Y el segundo, de 3×9 bits (BanReg-BU y BanReg-BY respectivamente), está conecta a la salida del buffer circular y a la entrada del multiplexor (Multiplexor-U y Multiplexor-Y respectivamente). El primer banco se utiliza para realimentar la información almacenada

en las memorias RAM1 y RAM2, y el segundo para almacenar temporalmente cada una de las columnas de la ventana de vecindad que intervienen en el procesamiento. Este último banco se emplea también para segmentar el cauce de la sub-etapa S2 e incrementar la frecuencia de funcionamiento del circuito.

El multiplexor recibe en paralelo los datos almacenados en el banco de registros (BanReg-BU o BanReg-BY) y los envía secuencialmente a la unidad MAC (MAC-U o MAC-Y), donde son procesados junto con los operadores (B o A), almacenados respectivamente en las memorias ROM. La microarquitectura interna de la unidad MAC incluye un multiplicador de 18 bits y un acumulador de 48 bits configurados para operar en complemento a 2 a máxima precisión. La unidad MAC está internamente segmentada en dos etapas. Los resultados de las operaciones MAC son sumados junto con el operador de bias I en el Proceso-Salida (sub-etapa S3) usando un sumador de tres entradas de 32 bits. La salida de dicho sumador es truncada a 9 bits y acotada por un comparador que representa la función de activación de la Etapa $F(x)$. Las salidas de la Etapa Uij, Yij y la señal de habilitación de las salidas OE son almacenadas en tres registros de salida sincronizados con el reloj CLK.

Para sincronizar la Etapa se utilizan las señales relojes CLK y CST, y una unidad de control cuya salida ha sido representa en la figura 4.18-a. El reloj CLK se utiliza para sincronizar la carga y salida de los datos de las sub-etapas S1 y S3, mientras que el reloj CST proporciona los 17 ciclos de reloj que necesita la unidad de control para secuenciar el procesamiento de la sub-etapa S2.

La figura 4.18-b muestra la temporización de las principales señales de datos de la Etapa a lo largo de un ciclo de procesamiento completo. Para garantizar el funcionamiento de la Etapa, los principales requisitos temporales a satisfacer son los tiempos de *setup* de la sub-etapa S2 (Ts2) y de la sub-etapa S3 (Ts3). Para ello, tanto el tiempo Ts2 (entre los flancos ascendente 4 y 5 del reloj CST) como el tiempo Ts3 (desde el flanco 17 del reloj CST hasta el siguiente flanco del reloj CLK) deberán ser mayor que la suma del tiempo de propagación de los flip-flops de salida de la sub-etapa previa (Tpf), el tiempo de retraso en la conexión entre las sub-etapas (Tn1 y Tn2 para cada caso), el tiempo de *setup* de los flip-flops de entrada de la sub-etapa siguiente (Tsf) y el tiempo de clock-skew máximo de las señales de reloj CLK y CST (Tskk y Tskt respectivamente).



a) Señales de la unidad de control

b) Señales de datos internas de la sub-etapa S2

Figura 4.18: Secuencia de ejecución de las sub-etapa S2. (a) Señales de control generadas por la unidad de control. (b) Señales de datos de la sub-etapa S2.

4.8.2. Implementación en VHDL de bajo nivel y sincronización *Self-Timed*

Los objetivos de esta implementación son conseguir la máxima eficiencia en el uso de los recursos hardware de la FPGA y maximizar la velocidad de procesamiento de la Etapa. Para ello, el circuito de la figura 4.17 ha sido implementado utilizando una descripción en VHDL de bajo nivel en la que se instancian primitivas de nivel tecnológico y se usan técnicas de posicionamiento manual de componentes (RPM, *Relationally Placed Macros*). Para realizar los circuitos se han tenido en cuenta las características de los componentes tecnológicos de más bajo nivel de las FPGAs Virtex-4 de Xilinx, plataforma elegida para la implementación. En particular la de los bloques de memoria (BRAM), las unidades de procesamiento digital (DSP48) y otras primitivas básicas como por ejemplo: FDCPE (registros), LUT (*Look-up table*) , MUXF5 (multiplexores), BUFG (buffers de reloj), etc.

Reloj interno CST basado en un oscilador *Self-Timed*

La señal de reloj CST ha sido generada a partir de un oscilador/contador interno de naturaleza asíncrona. El oscilador se inicia cada nuevo flanco ascendente en el reloj principal CLK y genera los 17 ciclos de reloj que requiere la unidad de control de la Etapa, a continuación se detiene.

El oscilador está basado en un micropipeline *Self-Timed*(ST) diseñado a partir de un protocolo *handshake* de 4-fases, elegido frente al de 2-fases por su mayor robustez y fiabilidad ante perturbaciones externas (Sparso y Furber 2001). Dada la naturaleza asíncrona del oscilador, la frecuencia del reloj generada dependerá de la profundidad de la lógica de un circuito de retardo y de la tecnología de fabricación de la FPGA. La dependencia de la frecuencia de reloj con la tecnología podrá ser vista como una ventaja, desde el punto de vista de la estabilidad de funcionamiento, dado que la utilización de los mismos recursos tecnológicos, tanto para implementar el circuito de retardo como los componentes de la Etapa, provocará que disminuyan de los efectos perjudiciales que sobre la sincronización del circuito producen las fluctuaciones de temperatura y de tensión de alimentación.

En la figura 4.19-a se muestran los principales componentes del Oscilador-ST. En la parte izquierda aparece el flip-flop y la lógica combinacional encargada de iniciar el oscilador y sincronizar las fases de los relojes CST y CLK. En el centro de la figura se aprecian los módulos handshake, basados en puertas C-MULLER de 4-fases (Ortega et al., 2003; Furbe y Day 1992), y el elemento de retardo formado por varias unidades de retraso idénticas (a mayor cantidad de unidades de retraso menor frecuencia de oscilación). En la parte superior de la figura se muestra el contador/comparador ascendente encargado de detener al oscilador una vez generados 17 pulsos.

La figura 4.19-b muestra el diagrama de sincronización de las señales de reloj CLK y CST. Como se observa, el Oscilador-ST inicia la generación de pulsos un tiempo (Tkt) después de que se produzca el flanco ascendente del reloj CLK. Dicho retraso será aprovechado por la Etapa para retrasar la puesta en marcha de la unidad de control con respecto a la captura de datos de la entrada. El tiempo de retraso Tkt viene determinado por la lógica interna del oscilador, e incluye el tiempo de propagación del flip-flop, el retraso de la lógica combinacional (puertas, handshake y buffer global) y el retardo en las conexiones.

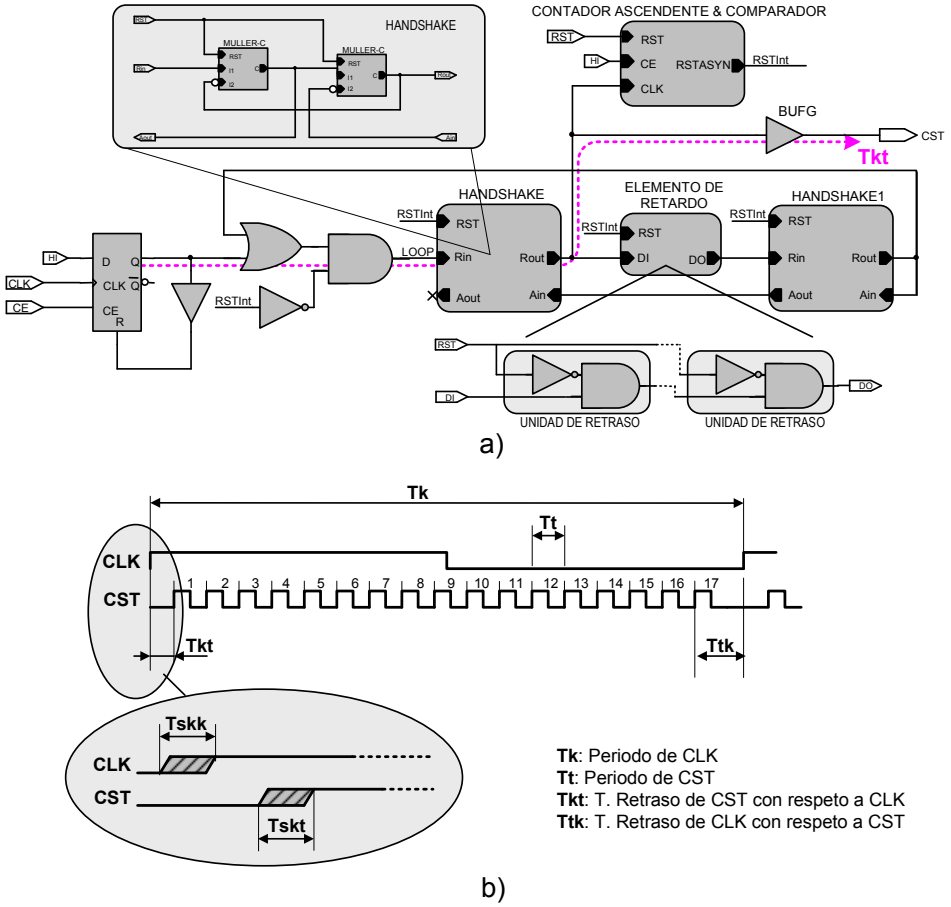


Figura 4.19: Oscilador-ST. a) Estructura interna del circuito Oscilador-ST. b) Sincronización de las señales de reloj CLK y CST.

Por otra parte, el tiempo que transcurre desde el último flanco ascendente del reloj CST (pulso 17) hasta el siguiente flanco ascendente del reloj CLK (Ttk) determinará el tiempo de *setup* de la sub-etapa S3 (Ts3). Dicho retraso podrá ser ajustado experimentalmente añadiendo o eliminado unidades de retraso al Oscilador-ST. Hay que indicar que el *clock-skew* de los relojes CLK y CST (Tskk y Tskt respectivamente) no será un problema en la sincronización de la Etapa, debido a su pequeño valor como consecuencia de la reducida ocupación de área de la Etapa y del uso exclusivo del osciladores.

La figura 4.20 muestra el posicionamiento y el rutado de los componentes tecnológicos de bajo nivel utilizados para implementar el Oscilador-ST. Como se aprecia en la figura, el oscilador hace uso de 14 Slices que han sido distribuidos sobre 4 columnas diferentes. Los bloques *Handshakes* y las dos unidades de retraso

han sido colocados en la columna de la izquierda, ocupando un total de 3 Slices. La lógica de inicio y sincronización de los relojes se sitúa a continuación, en la siguiente columna, y ocupan un solo Slice. Finalmente, el contador/comparador ascendente, con una profundidad lógica de 4 LUTs, ocupa de 10 Slices y ha sido situado en la tercera y cuarta columnas.

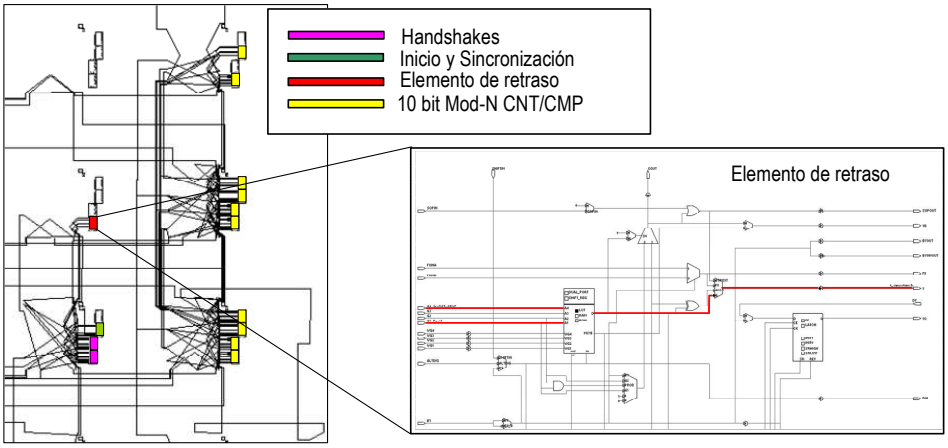


Figura 4.20: Distribución de los Slices utilizados por el oscilador ST.

El Oscilador-ST ha sido simulado y analizado experimentalmente con el fin de verificar su funcionamiento y determinar la relación entre la frecuencia real de oscilación y la obtenida por simulación. Las medidas realizadas con el osciloscopio han mostrado que la frecuencia real de oscilación, independientemente del número de elementos de retraso, es aproximadamente un 33 % superior a la que se obtiene en simulación. Esta desviación puede ser achacada al uso de parámetros temporales conservadores en las herramientas de simúlación, que por defecto vienen configuradas con los máximos valores de retardo de la tecnología.

Componentes de bajo nivel de la Etapa

La Etapa Carthago (figura 4.17) ha sido implementada sobre una FPGA XC4LX25-11 (Xilinx, 2004) instanciando primitivas de bajo nivel y realizando el posicionamiento de los componentes de forma manual. El objetivo de este proceso tan laborioso es conseguir la mejor relación área/velocidad del circuito. Esta metodología de diseño permite que los segmentos del pipeline de la sub-etapa S2 puedan tener una profundidad de lógica constante, fijada en 4 niveles de ló-

gica (LUTs), y que el retraso en las conexiones de los componentes puedan ser controlados mediante su colocación manual. Bajo estas condiciones, el número de elementos de retardo óptimo del Oscilador-ST podrá ser calculado en función de la profundidad máxima de la lógica (4 LUTs) y el retraso máximo en las conexiones de los componentes más alejados.

La figura 4.21 muestra la distribución de los componentes tecnológicos utilizados para implementar la Etapa Carthago. El circuito ha sido sintetizado con la herramienta XST (*Xilinx Synthesis Technology*) y el posicionamiento de los componentes ha sido supervisado con el editor gráfico *FPGA-Editor*, ambas herramientas incluidas en el entorno de desarrollo Xilinx ISE 10.1.

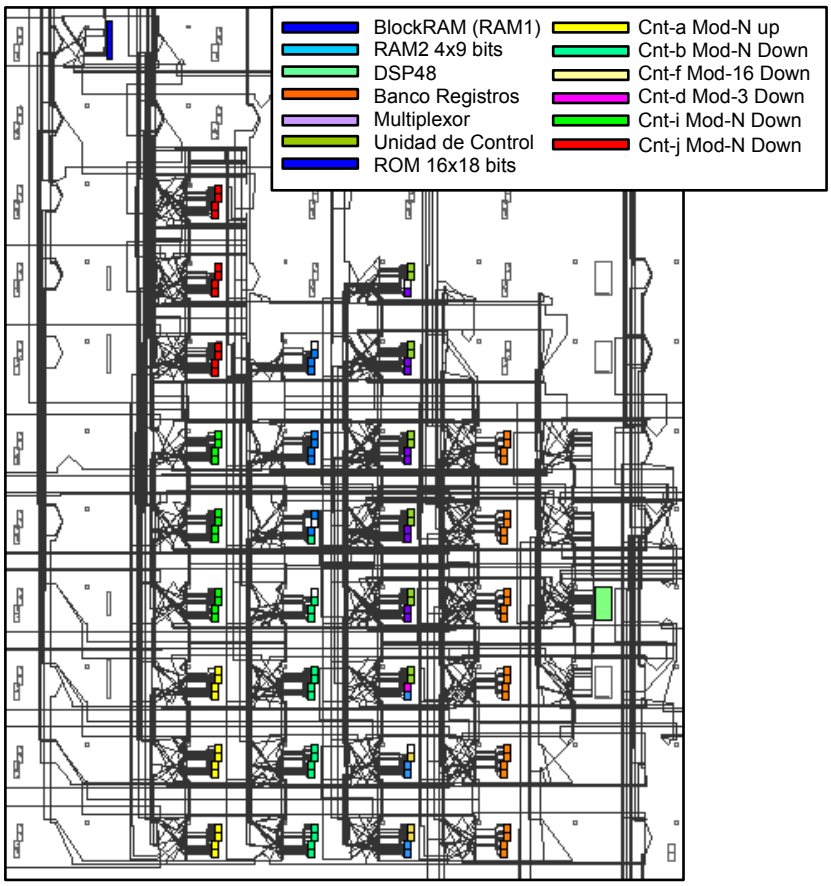


Figura 4.21: Distribución manual de los componentes principales de la Etapa Carthago.

Las memorias ROM asíncronas, utilizadas para almacenar los operadores de la plantilla (A y B), han sido implementadas como memoria distribuida usando

los Slices de la FPGA. El tamaño de palabra, y por consiguiente la precisión de los coeficientes, ha venido determinado por el tamaño de las entradas de 18 bits las unidades MAC, implementadas con primitivas DSP48. Las RAM1, utilizada por los buffers circulares, han sido implementadas usando primitivas BRAM configuradas en modo doble puerto y tamaño de 1024×18 bits. Las memorias RAM2, de escritura síncrona y lectura asíncrona, han sido implementadas de forma distribuida utilizando los Slices de la FPGA.

Los contadores del circuito han sido implementados usando primitivas del tipo: FDCPE, LUT4 y LUT3, eligiendo en cada caso la configuración ascendente o descendente más favorable desde el punto de vista del consumo de recursos hardware. Gracias al diseño manual de estos componentes, la restricción de máxima profundidad de lógica (4 LUTs) pudo ser satisfecha para el caso de los contadores de mayor tamaño (10 bits).

Los multiplexores han sido implementados con primitivas: LUT4 y MUXF5, registrando sus salidas para segmentar el circuito y conseguir mayor frecuencia de funcionamiento. Todos los registros utilizados por la Etapa han sido implementados con primitivas FDCPE, incluidas en los Slices de la FPGA. Por otra parte, los DSP48 utilizados por las unidades MAC han sido segmentados internamente en dos etapas para incrementar la frecuencia de funcionamiento de la Etapa. Finalmente, la unidad de control de la Etapa ha sido implementada con lógica combinacional usando primitivas LUT4 y un contador ascendente de 5 bits.

Resultados de la implementación

El circuito de la Etapa ha sido simulado y analizado con las herramientas *ModelSim* (*Mentor Graphics*) y *Timing Analyzer* (Xilinx ISE 10.1) con el fin de verificar su funcionamiento y determinar los caminos críticos del circuito. Para realizar las simulaciones, dichas herramientas fueron actualizadas con los últimos modelos temporales de las primitivas tecnológicas de la FPGA.

Las simulaciones temporales del circuito revelaron la existencia de dos caminos críticos principales: uno localizado en las conexiones que unen las sub-etapas S1 y S2, y el otro en las conexiones que unen las sub-etapas S2 y S3. En concreto, el primer camino crítico aparece entre la BRAM y los registros conectados a su salida. Y el segundo entre el DSP48 y los registros de segmentación colocados en la

entrada de la sub-etapa S3. La frecuencia máxima de funcionamiento de la Etapa viene determinada por estos caminos críticos y puede ser calculado sumando el tiempo de respuesta de las primitivas (BRAM o DSP48), el tiempo de retraso en la conexión con los registros, el tiempo *setup* de los registros y el *Clock-Skew* máximo de ambos relojes.

En la mayoría de las simulaciones realizadas, el camino crítico más desfavorable apareció entre la salida de la BRAM y los registros de entrada de la sub-etapa S2, puesto que el tiempo de respuesta de la BRAM es mayor que el del DSP48. Sin embargo, en general, el camino crítico del circuito dependerá de la conexión más lenta entre las dos primitivas indicadas y los correspondientes registros a los que se conectan. Tras realizar varias simulaciones sobre la FPGA XC4LX25-11, se comprobó que el mayor retraso en estos caminos crítico es de 2.9 ns.

Las simulaciones temporales del circuito han revelado, por tanto, que la Etapa puede funcionar a una frecuencia máxima de 344 MHz. Sin embargo, pruebas experimentales, realizadas para comprobar la máxima frecuencia real de funcionamiento, han demostrado que la Etapa puede operar correctamente y de forma estable a una frecuencia máxima de 415 MHz (máxima frecuencia alcanzada por el Oscilador-ST). Bajo estas condiciones, y teniendo en cuenta las dos entradas del circuito, la Etapa será capaz de procesar $2 \times (415/17) = 48.8$ MPS (Millones de Píxeles por Segundo). Lo que significa que, para unas entradas codificadas con 8 bits, el circuito tendrá un ancho banda constante de 48.8 MB/S (Mega Bytes por Segundo) y, por tanto, la posibilidad de trabajar con imágenes de 640×480 píxeles y 256 niveles de gris a $(415e6/17)/(640 \times 480) = 79$ fps (imágenes por segundo). Por otro lado, considerando que para procesar un píxel son requeridas 35 operaciones aritméticas (18 multiplicaciones y 17 sumas, incluyendo la suma del bias), en las condiciones indicadas anteriormente, la Etapa realizará un total 0.85 GOPS (Giga Operaciones por Segundo).

En cuanto al consumo de área del circuito, las primitivas DSP48 han demostrado ser los recursos más críticos del circuito, siendo necesario utilizar el 4 % de estas primitivas para poder implementar una sola Etapa en la FPGA XC4LX25. En estas condiciones, será posible implementar un máximo de 24 Etapas en dicha FPGA, lo que supone poder emular aproximadamente 7.3 millones Celdas procesando video (640×480 píxeles) en tiempo real (considerando Celdas de una sola Etapa organizadas en 24 Capas diferentes).

El cuadro 4.1 resume los resultados de la implementación de diferente número de Etapas sobre la FPGA XC4LX25-11. Como se observa, el consumo de recursos hardware crece proporcionalmente al número de Etapas, mientras que la frecuencia de funcionamiento del circuito disminuye conforme se incrementa el número de Etapas. La disminución en la frecuencia es debida a la rigidez de la distribución de los componentes de la Etapa y a las dificultades que encuentra la herramienta de síntesis para colocar los registros de la Etapa cerca de las primitivas DSP48 o BRAM a las que se conectan.

Cuadro 4.1: Resultados de la implementación de la Etapa Carthago en VHDL de bajo nivel. Datos referidos a la FPGA XC4LX25-11.

Carthago VHDL-Bajo nivel

XC4LX25-11 (%Uso)	1-Etapa	2-Etapa	4-Etapa	8-Etapa	24-Etapa
Slices	302(2 %)	604(5 %)	1208(11 %)	2416(22 %)	7248(67 %)
Flip flops	362(1 %)	724(3 %)	1448(6 %)	2896(13 %)	8688(40 %)
DSP48	2(4 %)	4(8 %)	8(16 %)	16(33 %)	48(100 %)
BlockRAM	2(2 %)	4(5 %)	8(11 %)	16(22 %)	48(66 %)
BUFGs	2(6 %)	3(9 %)	5(15 %)	9(28 %)	25(78 %)
Nº ciclos CST/píxel	17	17	17	17	17
F. Max. CST (sim/exp MHz)	344/415	344/415	343/415	273/-	233/-
F. Max. CLK (sim/exp MHz)	20.2/24.4	20.2/24.4	20.2/24.4	16.0/-	13.7/-
GOPS (sim/exp)	0.70/0.85	1.41/1.70	2.82/3.41	4.49/-	11.5/-

4.8.3. Implementación en VHDL a nivel RTL

La Etapa Carthago ha vuelto a ser implementada en la FPGA usando una descripción estructural a nivel RTL y dejando que la herramienta ISE realice la síntesis y el posicionamiento automático de todos los componentes. En esta implementación, el Oscilador-ST ha sido sustituido por un circuito basado en primitivas DCM (Digital Clock Manager), el cual sintetiza la señal de reloj CST a partir del reloj principal CLK. En este caso, la frecuencia del reloj CST no dependerá de elementos de retardo sino que lo hará de la frecuencia del reloj CLK y de los parámetros de configuración de los DCMs. Dado que los DCMs son recursos limitados, se ha optado por generar una única señal reloj CST que será compartida por todas las Etapas implementadas en la FPGA.

Reloj interno CST basado en primitivas DCM

En la figura 4.22 se muestra el circuito utilizado para generar la señal de reloj CST a partir del reloj principal CLK. El circuito está compuesto por tres primitivas DCM conectadas en cascada. El DCM-1 multiplica por 2 la frecuencia de oscilación de la señal CLK, mientras que los DCM-2 y DCM-3 multiplican por 3 sus correspondientes frecuencias de entrada.

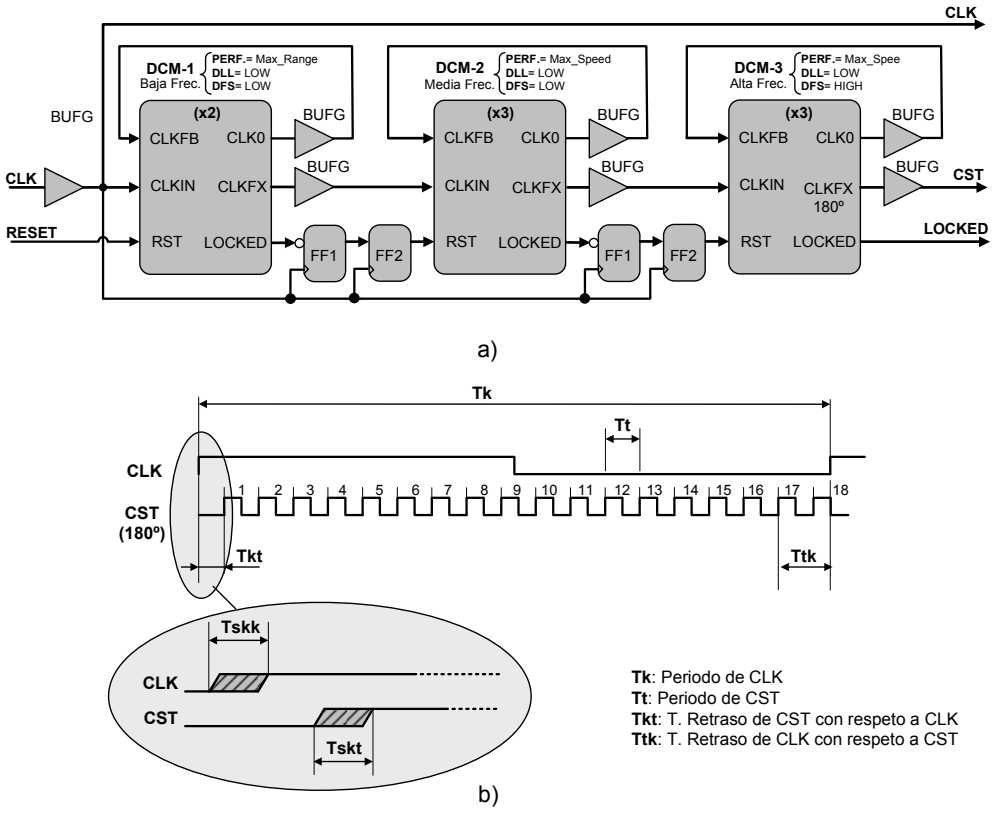


Figura 4.22: Generador de la señal de reloj CST basado en primitivas DCMs.

Tras un periodo transitorio, monitorizado por la salida LOCKED, el DCM-3 proporcionará de forma estable la señal de reloj CST, con una frecuencia 18 veces superior a la de la señal CLK y desfasada con ésta 180º. Tanto el factor de multiplicación como el desfase de la señal CST ha sido ajustada mediante los atributos de configuración internos de los DCMs. Hay que indicar que la frecuencia del reloj CLK ha sido multiplicado por 18 en lugar de por 17 (número de pulsos requerido por la unidad de control de la Etapa) debido a que 17 es un número primo y, por tanto, no es divisible entre varios DCMs.

La razón por la que se han utilizado tres DCMs en lugar de uno se debe a las propias limitaciones de estas primitivas para sintetizar frecuencias de salida muy diferentes a las de su entrada. Los DCMs de la familia Virtex-4 ofrecen múltiples modos funcionamiento que pueden ser agrupadas en tres categorías diferentes: modo baja frecuencia, modo media frecuencia y modo alta frecuencia. Para FPGAs Virtex-4 con grado de velocidad -11, el modo baja frecuencia soporta frecuencias de entrada entre [1-32] MHz y frecuencias de salidas en el rango [19-36] MHz. En el modo media frecuencia, la entrada debe estar comprendida en el rango [1-210] MHz, mientras que la salida estará comprendida en el rango [32-210] MHz. Finalmente, en el modo alta frecuencia, la entrada estará comprendida entre [50-315] MHz y la salida entre [210-315] MHz. Teniendo en cuenta lo anterior, para conseguir la máxima frecuencia de salida a partir de la mínima frecuencia de entrada es necesario utilizar al menos tres DCMs conectados en cascada. El primer DCM estará configurado en modo baja frecuencia, el segundo en modo media frecuencia y el tercero en modo alta frecuencia. Dichos modos de operación pueden ser configurados utilizando los siguientes atributos de los DCM: *Performance-mode*, *DLL-mode* y *DFS-mode*.

En la figura 4.22-b se muestra el cronograma de las dos señales de reloj generados por el oscilador y utilizadas por la Etapa. Como se observa, la señal de reloj CST presenta un desfase de 180º con respecto a la señal de reloj CLK. Este desfase supone el retraso (Tkt) necesario para garantizar la carga de datos en la Etapa antes de que la unidad de control inicie la secuencia de procesamiento. Por otra parte, el ciclo adicional de la señal de reloj CST (pulso 18), consecuencia de multiplicar por 18 la frecuencia del reloj CLK, proporciona el retraso suficiente (Ttk) para garantizar el tiempo de *setup* de la sub-etapa S3. El uso de Buffer Globales (BUFG), primitivas diseñadas específicamente para distribuir las señales de reloj en el interior de la FPGA, minimiza los problemas de *clock-skew* en el circuito y permiten que la Etapa pueda alcanzar altas velocidades de funcionamiento.

Resultados de la implementación

En el cuadro 4.2 se resumen los resultados de la implementación de la Etapa Carthago sobre la FPGA XC4LX25-11, a partir de la descripción en VHDL a nivel RTL estructural. Como se puede observar, el consumo de recursos hardware

presenta un incremento proporcional al número de Etapas, aunque inferior al obtenido en la implementación de bajo nivel (cuadro 4.1). Esto es debido a que, en este caso, las Etapas no incluyen el Oscilador-ST (como se puede apreciar en el menor consumo de primitivas BUFGs) y a que la herramienta de síntesis consigue mejorar el rendimiento de utilización de los Slices al poder entremezclar los componentes de diferentes Etapas. En lo que respecta al consumo de las primitivas DSP48 (que sigue siendo el recurso más crítico), el número de elementos sigue siendo el mismo que el obtenido en la implementación de bajo nivel, por lo que únicamente podrán implementarse 24 Etapas.

Cuadro 4.2: Resultados de la implementación de la Etapa Carthago en VHDL estructural a nivel RTL. Datos referidos a la FPGA XC4LX25-11.

Carthago VHDL-RTL

XC4LX25-11 (%Uso)	1-Etapa	2-Etapa	4-Etapa	8-Etapa	24-Etapa
Slices	200(1 %)	396(3 %)	793(7 %)	1591(14 %)	4767(44 %)
Flip-Flops	227(1 %)	454(2 %)	908(4 %)	1816(8 %)	5448(25 %)
DSP48	2(4 %)	4(8 %)	8(16 %)	16(33 %)	48(100 %)
BlockRAM	2(2 %)	4(5 %)	8(11 %)	16(22 %)	48(66 %)
BUFGs	2(6 %)	2(6 %)	2(6 %)	2(6 %)	2(6 %)
Nº ciclos CST /píxel	18	18	18	18	18
F. Max. CST (cir/DCM MHz)	363/315	344/315	344/315	339/315	310/315
F. Max. CLK (cir/DCM MHz)	20.1/17.5	19.1/17.5	19.1/17.5	18.8/17.5	17.2/17.5
GOPS (sim/exp)	0.70/0.61	1.33/1.22	2.67/2.45	5.27/4.90	14.46/14.7

Los resultados temporales de la tabla muestran que para un número de Etapas reducido, la frecuencia máxima de funcionamiento es similar a la obtenida en la implementación de bajo nivel. No obstante, a partir de 8 Etapas, la implementación estructural a nivel RTL presenta mejores resultados. Esta mejora en la frecuencia es consecuencia del mayor grado de libertad de la herramienta de síntesis para distribuir los componentes de la Etapa sobre la FPGA y, por tanto, para minimizar el retardo de los caminos críticos. Hay que indicar también que a pesar de que la herramienta de implementación informa de que el circuito puede funcionar a mayor frecuencia, la frecuencia máxima teórica del circuito vendrá limitada por la máxima frecuencia de funcionamiento de los DCM de 315 MHz. No obstante, considerando la frecuencia más ventajosa, las 24 Etapas implementadas en la FPGA podrán ejecutar un total 14.46 GOPS, lo que supone un resultado

ligeramente superior al obtenido por la implementación de bajo nivel. Para imágenes de tamaño 640×480 codificadas con 8 bits, una Etapa dispondrá de una ancho de banda de entrada de $2 \times (363/18) = 40.3$ MB/S y podrá trabajar a $(363e6/18)/(640 \times 480) = 65$ fps.

4.8.4. Implementación en Impulse-C sobre HPRC

La alternativa más rápida y sencilla para implementar el algoritmo de la red CNN es utilizar un PC convencional. Esta solución software, sin embargo, tiene el inconveniente de que consume demasiado tiempo para completar la ejecución. Por otra parte, una implementación hardware sobre FPGAs mejora las prestaciones de velocidad de la ejecución pero, sin embargo, exige mayor esfuerzo y tiempo de realización. Una solución intermedia que nos ha parecido interesante evaluar ha sido la implementación del algoritmo sobre computadores reconfigurables de altas prestaciones (HPRC, *High Performance Reconfigurable Computers*).

Esta nueva generación de computadoras ofrece como solución al problema de la aceleración de algoritmos un sistema híbrido formado por unidades de procesamiento estándar, basadas en microprocesadores de propósito general, y unidades de cómputo reconfigurable, basadas en FPGAs, estrechamente acopladas. Ejemplos de estos supercomputadores reconfigurable son el SRC-7, el SGI Altix 350 y el Cray XD1 (SRC-Computers, 2009; Silicon-Graphics, 2009; Cray, 2009). Estas plataformas tienen el potencial de aprovechar, simultáneamente, el paralelismo de grano grueso y grano fino de las aplicaciones, mostrando prestaciones en cuanto a rendimiento, potencia de cálculo y coste superiores a los computadores convencionales de alto rendimiento (HPC).

Con el fin de poder evaluar las prestaciones de la arquitectura Carthago sobre una plataforma HPRC, el algoritmo de la Etapa ha sido implementado sobre el servidor DS1002 de DRC Computer Corporation. Este computador está formado por un solo nodo que incluye un PC estándar y una unidad de procesamiento reconfigurable (RPU, *Reconfigurable Processor Unit*). El sistema DS1002 incorpora dos zócalos interconectados que contienen un microprocesador AMD Opteron Modelo 275 y una RPU110-L200. La RPU dispone de una FPGA Virtex-4 LX200, una memoria RAM DDR2 de 2GB y una memoria RLDRAM de baja latencia de 128 MB. La comunicación entre el procesador principal y el módulo RPU se

realiza mediante tres interfaces HyperTransport (HT), las cuales en esta plataforma están limitadas a 8 bits×400 MHz (DDR, *Double Data Rate*). El sistema de comunicación entre el microprocesador y la FPGA permite intercambiar información con un ancho de banda teórico de 800 MB/S por dirección (1.6 GB/S en ambas direcciones), lo que implica un rendimiento total de 4.8 GB/S si se consideran las 3 interfaces HT.

La figura 4.23 muestra la arquitectura del sistema DS1002 adaptada a nuestro problema. El algoritmo de la Etapa Carthago ha sido codificado en un proceso hardware, mientras que las tareas encargadas de leer las imágenes del disco duro, enviarlas a procesar, recibirlas y almacenarlas nuevamente en el disco duro, han sido combinadas y codificadas en un único proceso software denominado Productor-Consumidor. Para evaluar el rendimiento del sistema se utilizaron imágenes de test de 640 × 480 píxeles codificadas con 256 niveles de gris.

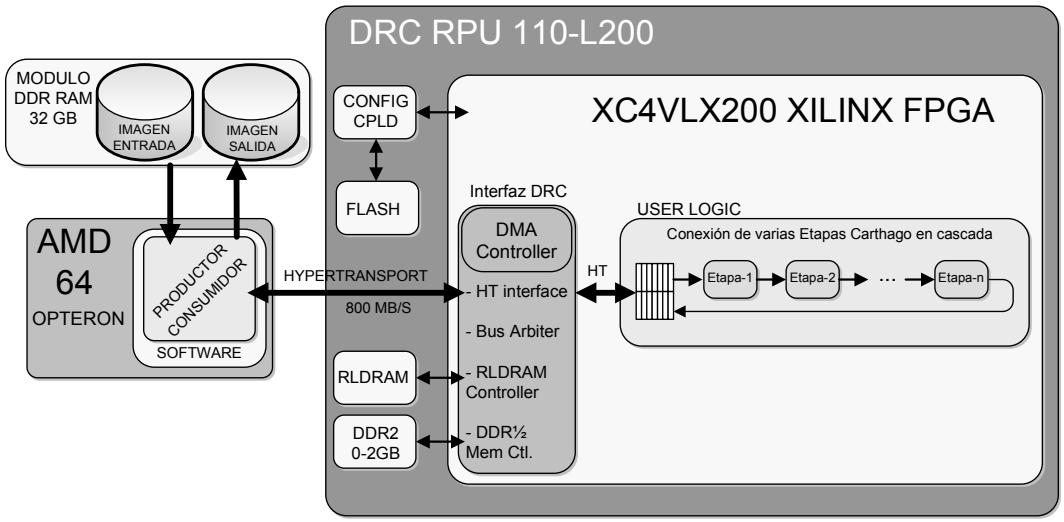


Figura 4.23: Diagrama de bloques del sistema DS1002 de DRC utilizado para implementar la Etapa Carthago.

Para analizar el consumo de recursos hardware y la velocidad de procesamiento del sistema se propusieron distintas implementaciones con diferente número de Etapas en cascada (1, 2, 4, 8 y 16). Todas las implementaciones fueron desarrolladas con la herramienta CoDeveloper de Impulse-C, realizando en cada caso dos versiones diferentes: una puramente software y otra compuesta por la parte software y la parte hardware. Todos las versiones fueron codificadas en lenguaje

C utilizando sintaxis ANSI C estándar y las funciones específicas de la API de Impulse-C, la cual incluye las rutinas de intercomunicación entre los procesos del sistema.

Las versiones puramente software fueron compiladas con el compilador estándar GCC, obteniéndose una aplicación que fue ejecutada sobre un PC convencional con Windows XP y sobre el DS1002 con Linux (Kubuntu 6.06 LTS). Las versiones hardware-software se separaron en un proceso software (Productor-Consumidor) y tantos procesos hardware como número de Etapas eran requeridas para la implementación. El proceso software fue compilado con el GCC y ejecutado en el microprocesador Opteron, mientras que los procesos hardware, compilados en VHDL y sintetizados con la herramienta ISE de Xilinx, fueron utilizados para configurar la RPU. Tanto el proceso software como los procesos hardware fueron ejecutados simultaneamente en el DS1002.

Resultados de la implementación

El cuadro 4.3 resume los resultados de tiempos y de consumo de área de las diferentes versiones hardware-software implementadas sobre el HPRC. Como se observa, el consumo de recursos lógicos de las diferentes realizaciones abarcan desde los 760 Slices, en la implementación de una sola Etapa, hasta los 11201 en el caso de la implementación de 16 Etapas, muy superior al obtenido por cualquiera de las dos implementaciones en VHDL anteriores.

Hay que aclarar que en la implementación sobre el DS1002, el algoritmo de la Etapa Carthago fue modificado para utilizar tres multiplicadores por unidad MAC y aumentar así la velocidad de procesamiento. Este cambio conlleva un mayor consumo de primitivas DSP48, lo que incide en que sea el recurso más crítico para la implementación. En este caso, es necesario el 100 % de los DSP48 de la FPGA V4LX200 para implementar la versión de 16 Etapas. El consumo de primitivas BRAM también se incrementa, en comparación con las alternativas en VHDL, debido a los 2 BRAM adicionales que han sido utilizados para implementar los canales de comunicación entre los procesos (un *Buffer-stream* por flujo de datos).

Otro parámetro interesante, el número de ciclos de reloj necesarios para procesar un píxel, muestra una perfecta correspondencia entre las estimaciones realizadas por la herramienta de depuración *debugger* de CoDeveloper y las medidas

experimentales obtenidas durante la ejecución. Considerando 6 multiplicadores por Etapa, el sistema requiere un total de 27 ciclos de reloj para completar el procesamiento de un píxel, valor que es independiente del número de Etapas implementadas.

Cuadro 4.3: Resumen de los recursos utilizados por diferentes implementaciones de la Etapa Carthago en Impulse-C sobre la plataforma DRC-DS1002. Datos referidos a la FPGA V4LX200-11 (incluida en la RPU).

Carthago Impulse-C (Unid. MAC con 3 Mult.)

V4LX200-11 (%Uso)	1-Etapa	2-Etapas	4-Etapas	8-Etapas	16-Etapas
Slices	760(1 %)	1455(1 %)	2847(3 %)	5657(6 %)	11201(12 %)
Flip-Flops	784(1 %)	1482(1 %)	2878(1 %)	5670(3 %)	11254(6 %)
DSP48	6(6 %)	12(12 %)	24(25 %)	48(50 %)	96(100 %)
BlockRAM	6(1 %)	10(2 %)	18(5 %)	34(10 %)	66(19 %)
Nº ciclos CLK/píxel	27	27	27	27	27
F. Max. CLK (MHz)	133	133	133	133	133
GOPS (sim/exp)	0.17	0.34	0.68	1.37	2.75

Tras la síntesis del circuito, se ha podido comprobar que el reloj más apropiado para el circuito, entre las opciones proporcionadas en el código de la RPU, es el de una frecuencia de 133 MHz, lo que implica un ancho de banda de entrada de las Etapas de $2 \times (133/27) = 9.8$ MB/S. Por otro lado, se ha podido comprobado también que la interfaz HT del sistema, configurada para transferir información a 64 bits y una frecuencia de 200 MHz, es capaz de soportar un ancho de banda de 1.6 GB/S (considerando ambas direcciones). En un seguimiento del código VHDL generado por Impulse-C se detectó que las unidades MAC inferidas por la herramienta de síntesis son implementadas como primitivas DSP48 no segmentadas, lo que influye en la baja frecuencia de funcionamiento del circuito.

La figura 4.24 muestra los tiempos de procesamiento de cada una de las versiones implementadas sobre las tres plataformas de cómputo propuestas: ejecución "sólo software" sobre un Core2Duo y Windows-XP, ejecución "sólo software" sobre el Opteron (DS1002) y Linux, y co-ejecución "hardware-software" sobre Opteron y FPGA (DS1002).

Las pruebas realizadas sobre imágenes de test de 640×480 píxeles y 8 bits de resolución han mostrado que la ejecución "sólo software", utilizando el Core2Duo

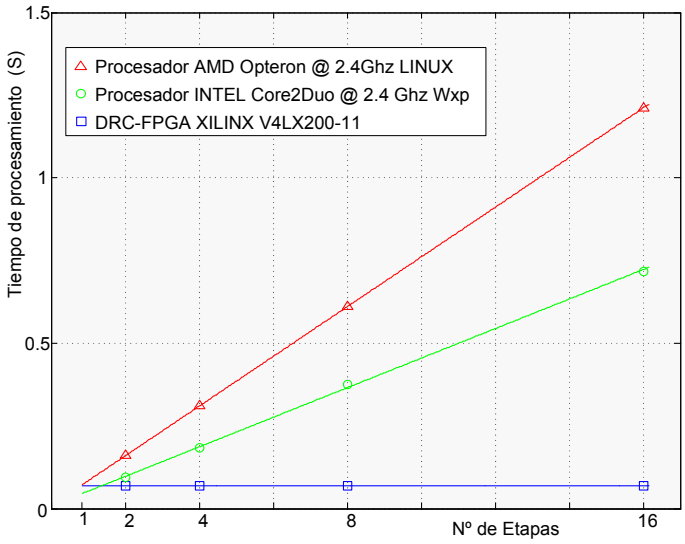


Figura 4.24: Tiempo de procesamiento de una imagen de 640×480 píxeles sobre las tres plataformas de utilizadas para la implementación de alto nivel.

a 2.4 GHz y Windows-XP, es la opción más rápida para la implementación de una sola Etapa, empleando en su ejecución 0.05 segundos. A partir de 2 Etapas, la co-ejecución "hardware-software" (sobre el DS1002) presenta mayor velocidad de procesamiento, manteniendo una marca de tiempo de 0.07s independiente del número de Etapas. El comportamiento lineal observado en la velocidad de procesamiento de las tres plataformas analizadas permite extrapolar los resultados a otras configuraciones distintas, formadas por diferente número de Etapas o distinta estructura neuronal.

Tras el análisis de los resultados de velocidad se ha podido comprobar, para el caso de una implementación con 16 Etapas e imágenes de tamaño 640×480, que el sistema DS1002 puede alcanzar una velocidad de procesamiento máxima de $(133e6/27)/(640 \times 480) = 16.03$ fps. Esto es 10.25 veces superior a la conseguida por un PC convencional (Core2Duo, de 2,4 GHz) en las mismas condiciones.

4.9. Etapa Carthagonova

La Etapa Carthagonova ha sido desarrollada a partir de la Etapa Carthago, manteniendo su mismo esquema de procesamiento pero modificando su arquitectura interna para incrementar su paralelismo. El objetivo de esta nueva ar-

quitectura es aumentar las prestaciones de velocidad de la Etapa y mejorar el aprovechamiento de los recursos internos de la FPGA.

Uno de los principales cambios incluidos en la Etapa Carthagonova ha sido la sustitución de los buffers circulares, utilizados por la arquitectura Carthago, por memorias con estructura tipo FIFO (*First In, First Out*). Estas memorias almacenan la misma información que los buffers circulares aunque tienen un modo de acceso más sencillo que agiliza el procesamiento y simplifica la arquitectura. Teniendo en cuenta que el flujo de datos que recibe la Etapa está organizado en filas y columnas ($M \times N$), la memoria FIFO ha sido diseñada para almacenar los $(2rN) + (2r + 1)$ últimos datos recibidos por la entrada, siendo r el radio de vecindad de la Etapa.

La figura 4.25 muestra el esquema interno de la memoria FIFO para el caso de una Etapa de radio de vecindad $r = 1$. La memoria FIFO dispone de una única entrada de datos y de $(2r + 1)$ salidas paralelas, por donde se obtienen las columnas de la ventana de vecindad a procesar.

Internamente, la FIFO está constituida internamente por una memoria RAM, con capacidad para $2rN$ datos, y por un conjunto de $(r+1)(r+2)/2$ registros, que almacenan los datos restantes y retrasan la información para que coincida en la salida. El direccionamiento de la memoria RAM se realiza mediante un contador ascendente de módulo N, denominado Contador-j, que sirve tanto de puntero de lectura y como de escritura. En cada posición de memoria, la FIFO almacena datos de diferentes filas de la matriz de entrada; en la parte menos significativa (fila inferior) se almacenan los datos de la última fila recibida, mientras que en la parte más significativa (fila superior) se almacenan los datos de la penúltima fila introducida, almacenados previamente en la parte menos significativa de la columna anterior.

Los procesos de escritura y lectura sobre la FIFO se realizan en el mismo flanco de reloj de forma síncrona. No obstante, cuando la FIFO recibe un nuevo dato (U_{65}, figura 4.25-a), antes de ser almacenado en memoria, en primer lugar se lee de la RAM la columna seleccionada por el Contador-j (j=5), que contiene tanto el dato que será realimentado a la memoria en la siguiente iteración (U_{55}, fila inferior) como el elemento superior de la columna de salida de la FIFO (U_{44}, fila superior). Tras la lectura, y en el mismo flanco, se procede a escribir en la

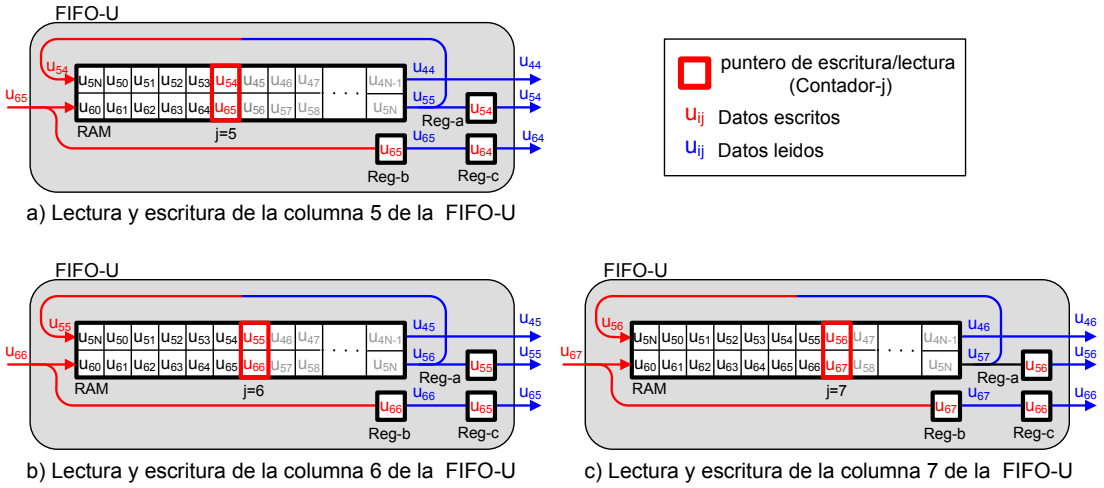


Figura 4.25: Funcionamiento interno de la memoria FIFO de la Etapa Carthagonova. Ejemplo para el caso de una Etapa de radio de vecindad 1 y matrices de entrada de tamaño $M \times N$.

memoria el dato de la entrada (U_{65}) y el dato que se realimenta en la actual iteración (U_{54}), ambos ocuparán la misma posición de memoria que acaba de ser leída (j=5). El dato de la entrada ocupará la parte menos significativa de la memoria (fila inferior), mientras que el dato realimentado se almacenará tanto en la parte más significativa de la memoria (fila superior) como en el registro Reg-a, que representa el elemento central de la columna de salida de la FIFO. Durante la fase de escritura, el contenido del registro Reg-b (U_{64}) pasará al registro Reg-c y el dato de la entrada (U_{65}) se copiará en el registro Reg-b. El registro Reg-c representa el elemento inferior de la columna de salida de la FIFO.

Las figuras 4.25-b,c representan los dos siguientes ciclos de lectura/escritura de la memoria FIFO (j=6 y j=7). A partir de este esquema, se puede deducir que dicha memoria introducirá una latencia de $Nr + 2r + 2$ ciclos en la Etapa, dado que para el dato de entrada U_{65} la ventana de vecindad obtenida a la salida estará centrada en el elemento U_{52}.

4.9.1. Algoritmo de la Etapa Carthagonova

En la figura 4.26 se representa el algoritmo de procesamiento completo de la Etapa Carthagonova. Como se observa, los datos de salida de las memorias

FIFO-U y FIFO-Y son llevadas a las matrices MatReg-U y MatReg-Y, respectivamente, para completar la ventana de vecindad que procesará la Etapa. Las matrices son de dimensión $(2r+1) \times (2r+1)$ y almacenan las $(2r+1)$ últimas columnas leídas de la FIFO. En cada iteración, el contenido de las matrices es desplazado una columna hacia la derecha para dejar hueco a la siguiente columna extraída de la FIFO.

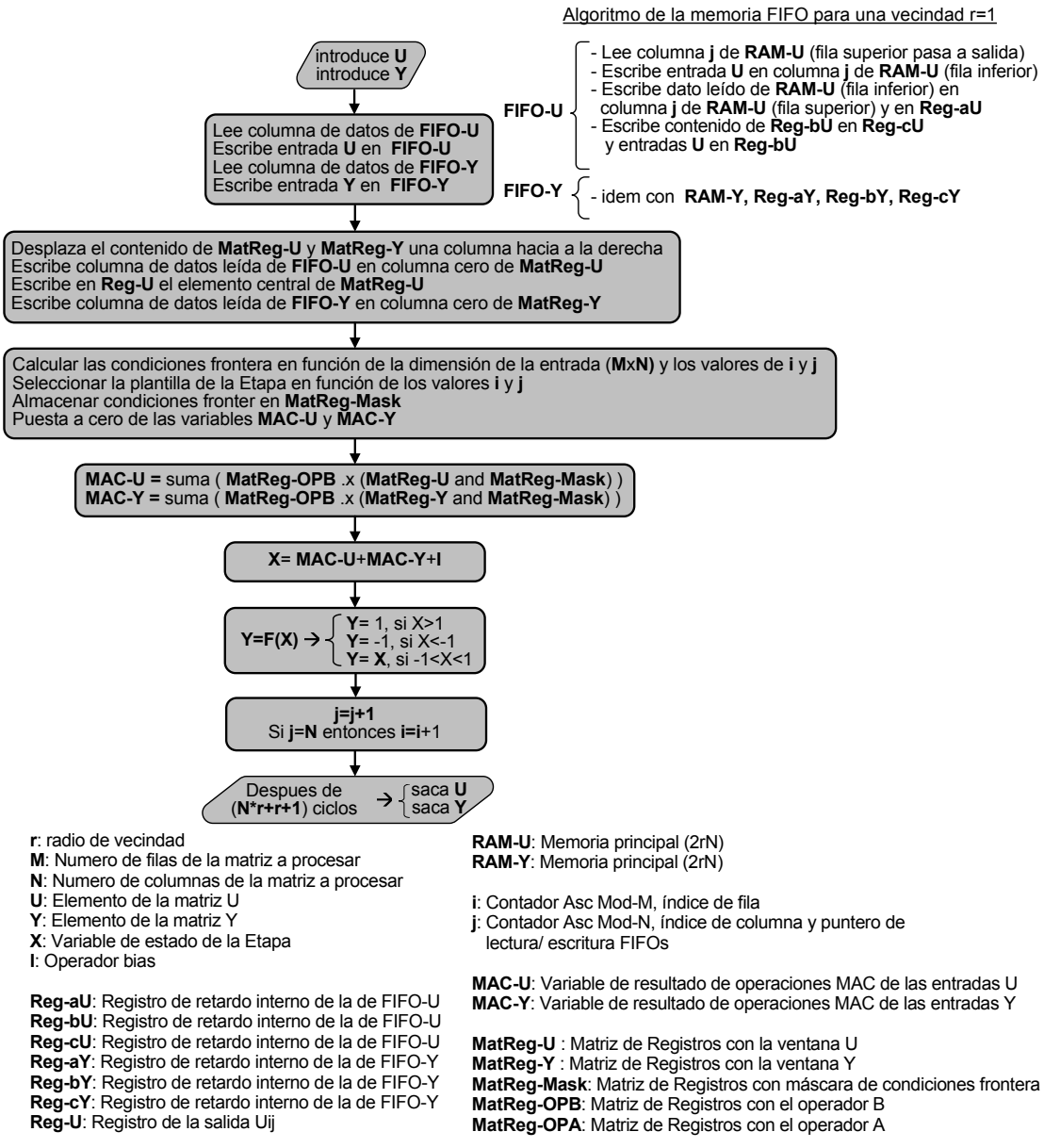


Figura 4.26: Algoritmo de la Etapa Carthagonova.

Las máscaras de fronteras de la Etapa son calculadas en cada iteración en función del tamaño de la matriz de la entrada ($M \times N$) y del valor de los contadores: Contador-i y Contador-j, los cuales determinan a que fila y columna de la matriz de entrada pertenece el actual dato recibido por la Etapa. Independientemente de que se den o no las condiciones frontera, las máscaras calculadas serán almacenadas en la matriz MatReg-Mask, de dimensión $(2r+1) \times (2r+1)$. Los valores de estas matriz (0 o 1) determinarán que elementos de las matrices MatReg-U y MatReg-Y tendrán que ser sustituidos por los valores de contorno de la Etapa (contorno Dirichlet). Los contadores: Contador-i y Contador-j se usan también para elegir la plantilla utilizada por la Etapa en cada iteración.

Una vez actualizadas las matrices: MatReg-U, MatReg-Y y MatReg-Mask, a continuación se realizarán las operaciones de multiplicación y acumulación (MAC) de la Etapa. Para ello, previamente, las mascarás frontera serán aplicadas a las matrices MatReg-U y MatReg-Y y el resultado multiplicado elemento a elemento con los operadores B y A de la plantilla. Dichos operadores deberán haber sido previamente almacenados en las matrices MatReg-OPB y MatReg-OPA, ambas de $NP \cdot (2r+1)^2$ elementos, siendo NP el numero de plantillas almacenadas y r el radio de vecindad de la Etapa. Conforme las operaciones son llevadas a cabo, los resultados serán acumulados en las variables MAC-U y MAC-Y, respectivamente.

Concluidas las operaciones MAC, los resultados de las variables MAC-Y y MAC-U serán sumados al operador de bias I, para obtener la variable de estado X de la Etapa y aplicarla como entrada a la función de activación $f(X)$. Una vez calculada la salida Y, el contador de columnas (Contador-j) se incrementará una unidad, mientras que el contador de fila (Contador-i) lo hará cuando el contador de columnas desborde su cuenta. Las salidas de la Etapa (Uij e Yij) serán habilitadas una vez haya transcurrido el tiempo de latencia de la FIFO ($Nr + r + 1$).

4.9.2. Paralelización de la Etapa Carthagonova

La figura 4.27 muestra el grafo de dependencias utilizado para paralelizar y adaptación al hardware el algoritmo de la Etapa Carthagonova. Al igual que sucedía en el caso de la Etapa Carthago, las tareas obtenidas en la descomposición han sido agrupadas en cuatro procesos paralelos: el Proceso-U, el Proceso-Y, el

Proceso-Salida y el Proceso-Control. El Proceso-U y el Proceso-Y tienen el mismo coste computacional y agrupan por separado las tareas de almacenamiento y procesamiento de los dos flujos de datos que recibe la Etapa: Uij e Yij. El Proceso-Salida recibe simultáneamente los resultados de los procesos anteriores y los sumará con el bias para generar la salida Yij de la Etapa. La tarea T10 se encarga de generar la salida Uij de la Etapa y representa la principal diferencia entre el Proceso-U y el Proceso-Y. El Proceso-Control, encargado de secuenciar el funcionamiento de la Etapa, agrupa las tareas compartidas relacionadas con el cálculo de las condiciones de frontera de la Etapa y con la generación de los valores de los contadores: Contador-i y Contador-j.

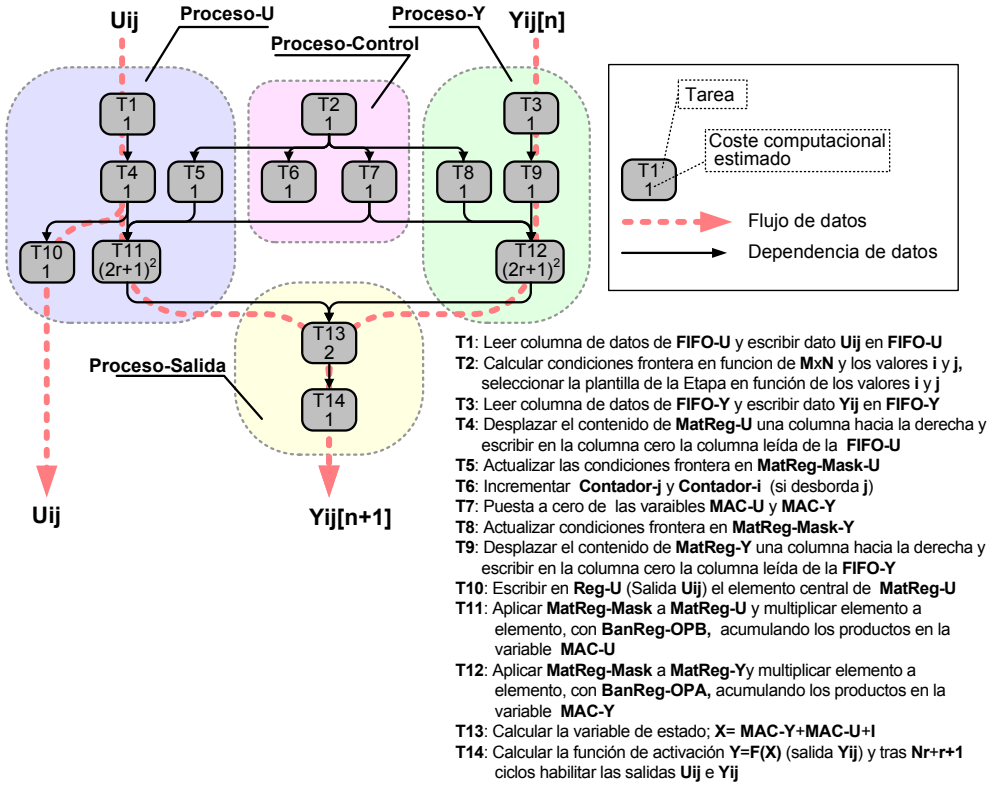


Figura 4.27: Paralelización de la Etapa Carthagonova: descomposición en tareas y agrupación en procesos.

Para llevar a cabo la descomposición del algoritmo se han seguido los mismos criterios que los considerados en el caso de la Etapa Carthago. Hay que destacar, no obstante, que los procesos de descomposición y agrupación del algoritmo han resultado más sencillos que en el caso anterior debido al menor tamaño del algo-

ritmo y a un grafo de dependencias más lineal y adecuado para la paralelización y adaptación al hardware.

4.9.3. Descripción de la Etapa Carthagonova a nivel RTL

La figura 4.28 muestra la arquitectura interna de la Etapa Carthagonova a nivel RTL, particularizada para entradas de datos de 9 bits y radio de vecindad $r = 1$.

Como se observa en dicha figura, las entradas U_{ij} e Y_{ij} de la Etapa se encuentran conectadas, respectivamente, a las memorias FIFO-U y FIFO-Y incluidas en los procesos Proceso-U y Proceso-Y. En dichos procesos se encuentran además las unidades de multiplicación y acumulación bidimensional (MAC-2D) de la Etapa, encargadas de procesar los datos extraídos de las memorias FIFO. En cada unidad MAC-2D, las columnas leídas de la FIFO son almacenadas en un banco de registro donde se completa la ventana de vecindad que se procesará en cada iteración. Las operaciones realizadas por cada unidad MAC-2D consisten en $(2r + 1)^2$ multiplicaciones y $(2r + 1)^2 - 1$ operaciones de suma. El elemento central de la ventana de vecindad del Proceso-U es almacenado temporalmente en el registro Reg-U para posteriormente sacarlo por la salida U_{ij} de la Etapa. Por otra parte, el Proceso-Control se encarga de comprobar las condiciones de frontera de la Etapa y secuenciar la ejecución interna de las unidades MAC-2D. El Proceso-Salida genera las salidas de la Etapa e incluye un sumador de tres entradas, para sumar los resultados de las unidades MAC-2D y el bias seleccionado, y un comparador que resuelven la función de activación de la Etapa.

Con el fin de incrementar la velocidad de procesamiento de la Etapa, la arquitectura del circuito ha sido segmentada en tres sub-etapas. La sub-etapa S1 representa la carga de datos en la Etapa e incluye los contadores Cnt-i y Cnt-j del Proceso-Control y el circuito combinacional que calcula los valores de las máscaras de fronteras. La sub-etapa S2 contiene la unidad de control y las unidades MAC-2D encargadas de llevar a cabo las principales operaciones de cálculo de la Etapa. Finalmente, la sub-etapa S3 representa la salida de la Etapa y tiene como tareas sumar los resultados proporcionados por la sub-etapa S2 con el bias y resolver la función de activación de la Etapa. Las sub-etapa S1 y S3 están conectadas a la región-CLK y, por tanto, las entradas y salidas de la Etapa tendrán el periodo

de muestreo de la señal del reloj principal CLK. La sub-etapa S2, está conectada a la región-CST/CLK, lo que significa que podrá estar sincronizada con el reloj CST o con el reloj CLK, en función de la configuración interna elegida para las unidades MAC-2D. Cuando la sub-etapa S2 utilice el reloj CLK, el circuito estará segmentado en un mayor número de etapa. Cuando la sub-etapa S2 utilice el reloj CST, el circuito estará super-segmentado.

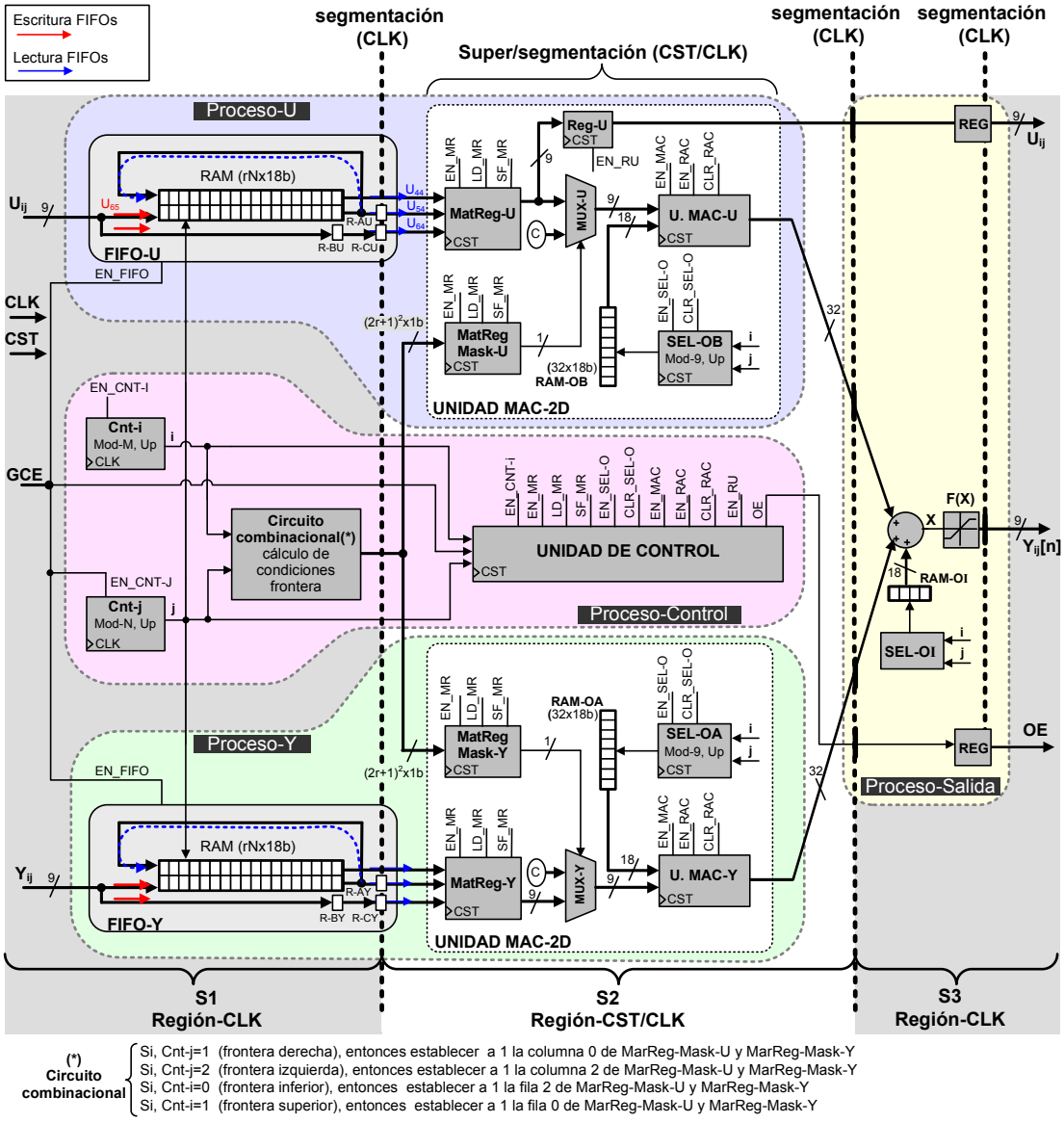


Figura 4.28: Descripción RTL de la Etapa Carthagonova para un radio de vecindad $r = 1$.

La conexión entre las sub-etapas S1 y S2 se realiza a través de las $2r + 1$ salidas de las memorias FIFO y las correspondientes entradas de los bancos de registros MatReg-U y MatReg-Y de las unidades MAC-2D. Cada uno de estos bancos disponen de $(2r + 1) \times (2r + 1)$ registros de 9 bits, con la capacidad para desplazar su contenido en varias direcciones diferentes. Durante la carga de los datos, los bancos son gestionados como un conjunto de $(2r + 1)$ filas de $(2r + 1)$ registros conectados en cascada, que desplazan su contenido una posición a la derecha cada nueva entrada de datos. Durante la lectura, el número de salidas de los bancos y la dirección del desplazamiento dependerá de la configuración interna de la unidad MAC-2D, como se verá en la sección siguiente.

Dentro de la unidad MAC-2D, la salida del banco de registros es llevada a un multiplexor que selecciona los datos que pasarán a la entrada de la unidad de multiplicación y acumulación (MAC-U o MAC-Y), es decir, decidirá si han de pasar los datos proporcionados por el banco de registros o si, por el contrario, han de pasar los valores de las condiciones de contorno (condición Dirichlet de valor configurable). El multiplexor estará controlado por la máscara binaria de fronteras (MatReg-Mask-U o MatReg-Mask-Y), cuyo contenido está asociado elemento a elemento con los datos del banco de registro. Los valores de las dos máscaras serán idénticos y serán calculados por el circuito combinacional del Proceso-Control. Las mascaras obtenidas serán función de los valores de los contadores Cnt-i y Cnt-j y de la dimensión de la matriz de entrada ($M \times N$). La otra entrada de la unidad MAC será cargada con los coeficientes de los operadores (B o A) de la plantillas seleccionada. Dichos operadores habrán sido almacenados previamente en las memorias asíncronas RAM-OB y RAM-OA, cuyo tamaño y profundidad dependerá del número de plantillas almacenadas, de la precisión de los coeficientes y de la configuración interna de las unidades MAC-2D. El direccionamiento de las memorias se realizará mediante un circuito selector (SEL-OB o SEL-OA) compuesto por un contador ascendente de modulo 9 y un circuito combinacional. El circuito selector utilizará los valores de los contadores Cnt-i y Cnt-j para cambiar las plantillas en función del espacio o en el tiempo. El número de multiplexores y la configuración interna de los bancos de registros dependerá de las características internas de la unidad MAC-2D utilizada.

Los resultados obtenidos por las dos unidades MAC-2D serán sumados al operador de bias I en el Proceso-salida (sub-etapa S3). El resultado de dicha suma

será utilizado como entrada del circuito comparador encargado de implementar la función de activación de la Etapa (salida Yij). Como se aprecia en la figura 4.28, los registros de entrada y salida del Proceso-salida han sido aprovechados para segmentar el cauce de la Etapa, lo que introducirá dos ciclos de latencia extra en el circuito. La latencia total del circuito vendrá determinada por la suma de los tiempos de latencia de la memoria FIFO ($Nr + r + 1$) ciclos, el introducido por la sub-etapa S2, que depende de la configuración interna de la unidad MAC-2D, y los dos ciclos extra que se introducen en la sub-etapa S3.

4.9.4. Unidades MAC-2D

La unidad MAC-2D es el componente clave para conseguir que la Etapa pueda alcanzar el máximo rendimiento en velocidad con el menor coste en recursos hardware. La arquitectura Carthagonova ha sido diseñada para utilizar tres tipos de unidad MAC-2D diferentes, cada una de ellas con distinto compromiso en cuanto a consumo de recursos de cómputo y velocidad de procesamiento. El objetivo es ofrecer distintas posibilidades a la hora de adaptar la arquitectura a los requisitos de diferentes aplicaciones. Con dicho propósito, las unidades MAC-2D han sido diseñadas para explotar a diferente nivel el paralelismo implícito de la arquitectura: la primera utiliza un solo multiplicador, la segunda tantos multiplicadores como filas tenga la ventana de vecindad a procesar y la tercera tantos multiplicadores como elementos tenga dicha ventana. En los siguientes apartados se analizan las características principales de cada una de estas unidades.

Unidad MAC-2D con un multiplicador

La unidad MAC-2D con un solo multiplicador será la opción más adecuada cuando los requisitos de velocidad no sean altos o cuando interese reducir al máximo el consumo de multiplicadores. En la figura 4.29-a se muestra la microarquitectura de ésta unidad para el caso de una Etapa de radio de vecindad $r = 1$ (ventana 3×3). En esta unidad, la lectura del banco de registros se realiza dato a dato, considerando que todos los registros están conectados en cascada y que su contenido se desplaza a lo largo de toda la cadena de forma secuencial. Para evitar la pérdida de datos, que impediría completar la siguiente ventana de ve-

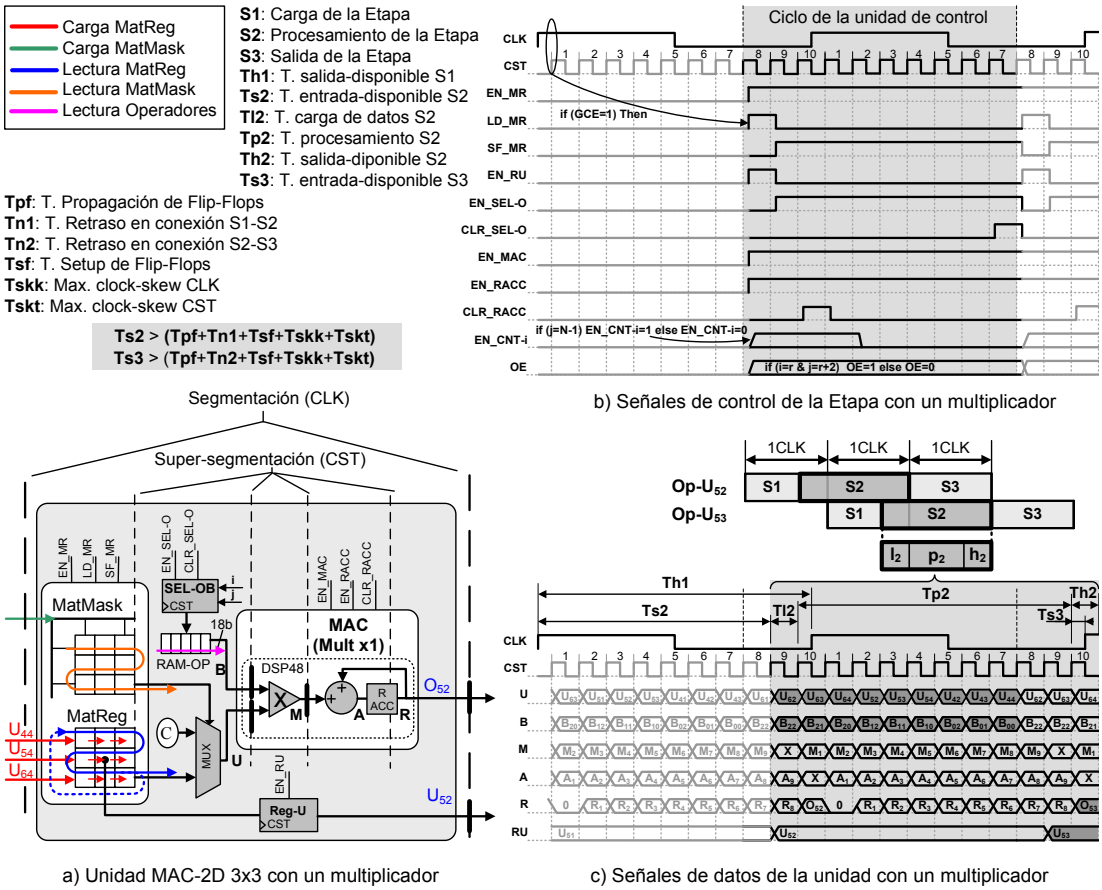


Figura 4.29: Microarquitectura y cronogramas de la unidad MAC-2D con un multipliacador para una Etapa de radio de vecindad 1.

cindad, el banco de registros ha sido configurado en lazo cerrado conectando la salida del último registro a la entrada del primero.

Conforme la ventana de vecindad es leída, los bits proporcionados por la máscara de fronteras, gestionada igualmente como un registro de desplazamiento serie, son utilizados para determinar, a través del multiplexor, si la salida del banco de registros ha de pasar a la entrada del multiplicador o lo ha de hacer la constante de contorno C. La otra entrada del multiplicador es alimentada con los coeficientes del operador B (o A), almacenados, previamente, en una memoria RAM de $NP \cdot (2r+1)^2$ palabras de 18 bits, siendo NP el número de plantillas almacenadas. El direccionamiento de esta memoria es realizado mediante un circuito selector gestionado por la unidad de control a través de los contadores Cnt-i y Cnt-j. Todos los componentes secuenciales de la unidad MAC-2D se encuentran

sincronizados con la señal de reloj CST. Dicha señal de reloj ha sido sintetizada a partir del reloj principal CLK, usando el circuito de DCMs de la figura 4.22.

El cauce de la unidad MAC-2D ha sido segmentado en cinco etapas, utilizando un total de $(2r+1)^2+3$ ciclos de reloj para completar el procesamiento de una ventana de vecindad de radio r. No obstante, dado que las operaciones se ejecutan de forma solapada, la unidad calculará una nueva salida cada $(2r+1)^2+1$ ciclos del reloj CST. Hay que indicar que uno de estos ciclos será utilizado para resetear el registro acumulador de la unidad MAC, y así poder iniciar las operaciones de acumulación de la siguiente ventana de vecindad.

El ciclo de reset podrá ser eliminado modificando la unidad MAC con la configuración mostrada en la figura 4.29-b. Sin embargo, habrá que tener en cuenta que esta configuración implica desaprovechar la mitad de las primitivas DSP48 de la familia Virtex-4, ya que, en dichas FPGAs, cada dos DSP48 comparten el mismo puerto de entrada externo (puerto C). En las familias Virtex-5, Virtex-6 y Series-7 no habría inconveniente en utilizar esta configuración. Sin embargo, sería interesante analizar que efectos tendría la relimentación externa de la unidad MAC sobre la frecuencia de funcionamiento del circuito.

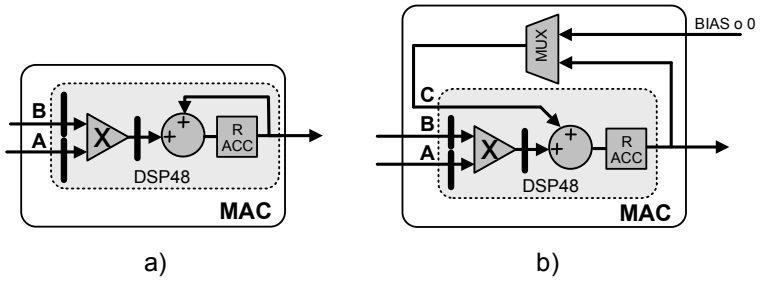


Figura 4.30: Unidades MAC implementadas con el DSP48. a) Esquema con realimentación interna. b) Esquema con realimentación externa.

Finalmente, las figuras 4.29-b y 4.29-c muestran, respectivamente, las señales de control de la unidad MAC-2D y el cronograma de sus principales señales datos, para el caso del procesamiento de la ventana de vecindad centrada en el elemento U_{53}. Como se aprecia en la figura 4.29-c, y a diferencia de lo que ocurre en la arquitectura Carthago, la conexión entre las memorias FIFOs y los registros de entrada de la sub-etapa S2 no representará un camino crítico en el circuito, debido a que el tiempo de disponibilidad de los datos de salida de la sub-etap S1 en la

entrada de la sub-etapa S2 (Ts2) ha sido incrementado considerablemente. En dicha figura se puede comprobar además que la unidad MAC-2D introduce un ciclo de latencia adicional (reloj CLK) en la respuesta de la Etapa.

Unidad MAC-2D con $(2r+1)$-multiplicadores

En aplicaciones con mayores requerimientos de velocidad, una posibilidad es incrementar el paralelismo interno de la unidad MAC-2D utilizando tantos multiplicadores como filas tenga la ventana de vecindad a procesar. Como se observa en la figura 4.29-a, esta arquitectura acelerará el procesamiento de la Etapa ejecutando $(2r+1)$ flujos de datos en paralelo, a costa, claro está, de un mayor consumo de recursos de cómputo, aproximadamente $(2r+1)$ veces superior al de la unidad MAC-2D con un solo multiplicador.

En este esquema, la lectura del banco de registros es gestionada de la misma forma que la carga, es decir, considerando al banco registros como un conjunto de $(2r+1)$ filas de $(2r+1)$ registros conectados en cascada que desplazan su contenido hacia la derecha. En este caso, para evitar la pérdida de datos en el desplazamiento, la salida del último registro de cada fila es realimentada a la entrada del primer registro de la misma fila. Las condiciones de frontera son controladas por los $(2r+1)$ multiplexores conectados a la salida de cada fila de registros y a la constante de contorno C. La máscara de fronteras presenta el mismo esquema de lectura que el utilizado por el banco de registros, es decir, considerando cada fila como un registro de desplazamiento serie, aunque en este caso sin realimentación. Conforme los registros de la máscara son leídos, los bits son utilizados como señal de selección de los multiplexores. Los coeficientes de los operadores A y B son cargados en la unidad MAC, desde la memoria RAM, como palabras de $(2r+1) \times 18$ bits.

Esta unidad MAC-2D ha sido super-segmentada en cinco etapas y requiere una frecuencia de reloj CST $(2r+2)$ veces superior a la del reloj principal CLK. Las figuras 4.31-b y 4.31-c muestran respectivamente, las señales de control utilizadas para secuenciar la unidad y el cronograma de las principales señales de datos, para el caso del procesamiento de la ventana de vecindad centrada en el elemento U_{53}. La latencia introducida por la unidad en la respuesta de la Etapa será de un ciclo de reloj CLK.

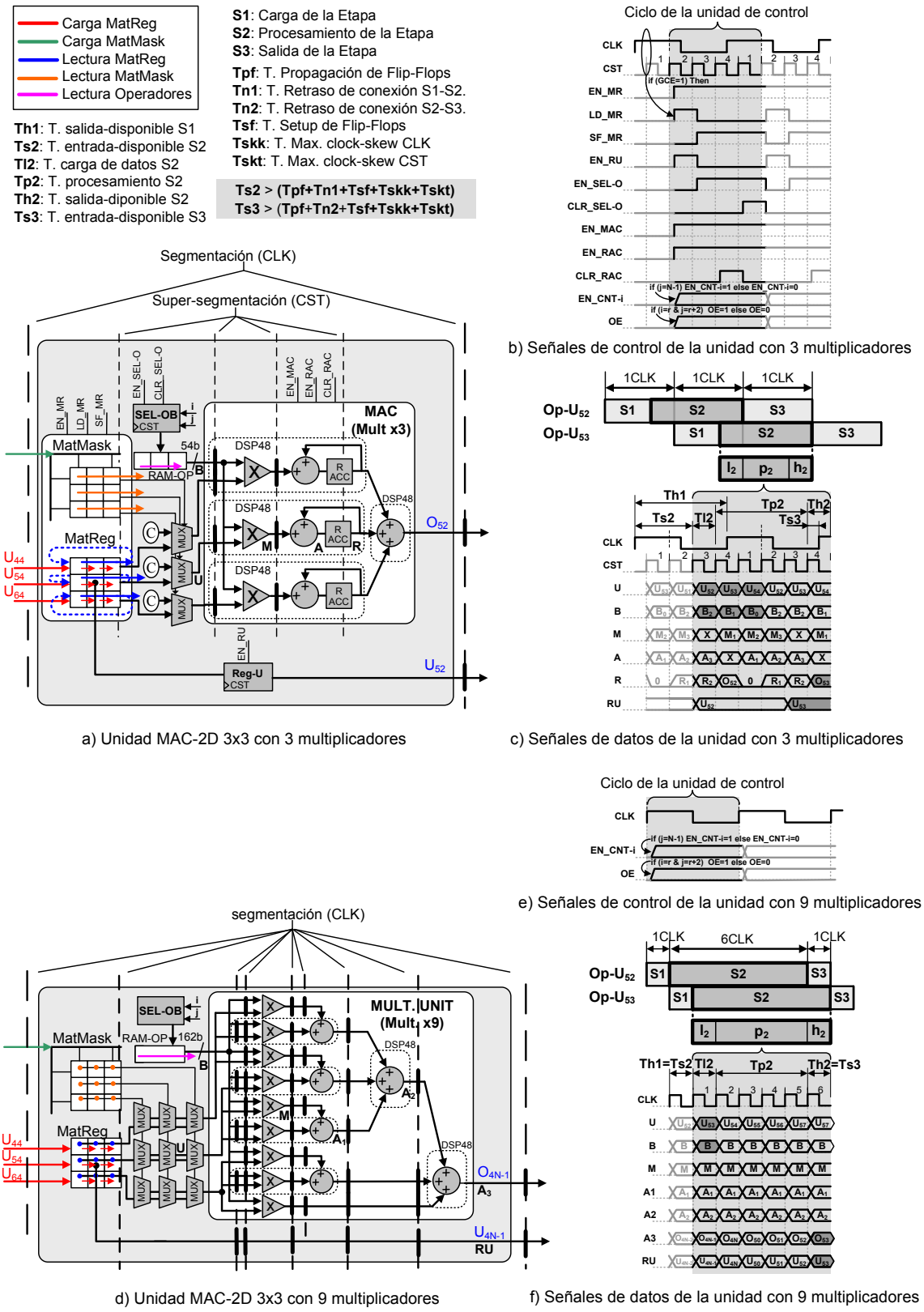


Figura 4.31: Descripción RTL de la Etapa Carthagonova para un radio de vecindad $r = 1$.

Unidad MAC-2D con $(2r+1)^2$-multiplicadores

La figura 4.29-d muestra la arquitectura de la unidad MAC-2D con las máximas prestaciones en cuanto a velocidad de procesamiento. En este caso, la unidad utiliza tantos multiplicadores como elementos tenga la ventana de vecindad a procesar. Con ello se consigue maximizar el paralelismo de la arquitectura y la velocidad de procesamiento, aunque a costa de un mayor consumo de recursos de cómputo. Bajo esta configuración, el banco de registros es considerado como un conjunto de $(2r+1)^2$ registros independientes que proporcionan $(2r+1)^2$ datos en paralelo a un conjunto de $(2r+1)^2$ de multiplexores, encargados de gestionar las condiciones de contorno de la Etapa y hacer llegar a los multiplicadores los datos a procesar. Obviamente para poder controlar independientemente las salidas de los multiplexores, la máscara de fronteras ha sido configurada como un conjunto de $(2r+1)^2$ registros de 1 bit, con sus salidas conectada a cada una de las las entradas de selección de los multiplexores.

A partir de esta configuración, los $(2r+1)^2$ multiplicadores podrán realizar todas las operaciones de multiplicación en un solo ciclo de reloj CLK. Lo que implica que todos los coeficientes de los operadores tendrán que ser leídos en paralelo y, por tanto, almacenados previamente como una única palabra de $(2r+1)^2 \times 18$ bits, lo que, por otro lado, permitirá eliminar el contador del circuito selector.

Esta unidad utiliza la señal de reloj principal CLK y ha sido segmentada en seis etapas, por lo que tras una latencia de 6 ciclos, proporcionará un nuevo resultado cada flanco del reloj. Las figuras 4.31-e y 4.31-f muestran las señales de control de esta unidad y sus principales señales de datos.

4.9.5. Implementación en VHDL a nivel RTL

Las tres versiones de la arquitectura Carthagonova han sido implementadas en VHDL estructural a nivel RTL, los únicos componentes instanciandos a bajo nivel han sido las primitivas DSP48. Las características de la descripción en VHDL utilizada permite que la arquitectura puede ser implementada en las siguientes familias de FPGAs de Xilinx: Virtex-4, Virtex-5 y Series-7.

Con el fin de evaluar las prestaciones de las arquitecturas, la implementación de la Etapa fue particularizada para las siguientes condiciones: radio de vecindad

1, matrices de entrada de dimensiones máximas 1024 × 1024 píxeles, entradas de datos de 9 bits, plantillas codificadas en punto fijo y 18 bits de precisión y operaciones con aritmética de punto fijo a máxima precisión (excepto la salida de la Etapa que es truncada a 9 bits).

Las memorias FIFOs son inferidas por la herramienta de síntesis como una primitiva BRAM de simple puerto y tres registros de 9 bits. En la BRAM se almacenan dos filas de datos completas, mientras que en los registros se almacenan los restantes datos recibidos por la Etapa. Los coeficientes de las plantillas son almacenados en una memoria RAM distribuida asíncrona. Y los multiplicadores y acumuladores de las unidades MAC-2D son implementados usando primitivas DSP48, configurados en cuatro modos diferentes (Xilinx, 2008): sumador de tres entradas, unidad de multiplicación, unidad de multiplicación y suma, y unidad de multiplicación suma y acumulación. Para llevar a cabo la operación de suma en la sub-etapa S3, las arquitecturas super-segmentadas utilizan un sumador descrito en VHDL de alto nivel, mientras que la arquitectura restante utiliza un DSP48 segmentado en dos niveles, dada su mayor velocidad.

Resultados de la implementación

Las descripciones en VHDL de las diferentes versiones de la arquitectura Carthagonova han sido sintetizadas con la herramienta XST de Xilinx ISE 10.1 para la FPGA XC4LX25-11. La comprobación del circuito y su análisis temporal han sido realizados con el simulador *ModelSim* y la herramienta *Timing Analyzer*. El cuadro 4.4 resume los resultados de utilización de recursos hardware y velocidad de funcionamiento en cada caso.

Las simulaciones realizadas han permitido determinar que el principal camino crítico de las arquitecturas está localizado entre las salidas de las unidades MAC-2D y la entrada de la sub-etapa S3. Para minimizar dicho camino, el fichero de restricciones de implementación fue configurado para favorecer la colocación de las primitivas DSP48 de la sub-etapa S2 lo más cerca posible de los registros de entrada de la sub-etapa S3. Con el mismo propósito, el número de ciclos del reloj CST fue incrementado una unidad para aumentar el tiempo de disponibilidad de la salida de la sub-etapa S2 en la entrada de la sub-etapa S3 (Ts3) (ver figuras 4.29-c y 4.31-c). Este ciclo extra, sin embargo, podría ser eliminado

Cuadro 4.4: Resultados de la implementación de las tres versiones de la Etapa Carthagonova en VHDL a nivel RTL. Datos referidos a una FPGA XC4LX25-11 de Xilinx.

Carthagonova VHDL-RTL (MAC-2D con 1 Mult.)

XC4LX25-11 (%Uso)	1-Etapa	2-Etapa	4-Etapa	8-Etapa	24-Etapa
Slices	264(2 %)	540(5 %)	1079(10 %)	2162(20 %)	6485(60 %)
Flip-Flops	357(1 %)	734(3 %)	1468(6 %)	2936(13 %)	8808(40 %)
DSP48	2(4 %)	4(8 %)	8(16 %)	16(33 %)	48(100 %)
BlockRAM	2(2 %)	4(5 %)	8(11 %)	16(22 %)	48(66 %)
BUFGs	2(6 %)	2(6 %)	2(6 %)	2(6 %)	2(6 %)
Nº ciclos CST/píxel	11	11	11	11	11
F. Max. CST (cir/DCM MHz)	438/315	438/315	437/315	430/315	353/315
F. Max. CLK (cir/DCM MHz)	39.8/28.6	39.8/28.6	39.7/28.6	39.0/28.6	32.0/28.6
GOPS (cir/DCM)	1.39/1.00	2.78/2.00	5.56/4.00	10.9/8.01	26.9/24.0

Carthagonova VHDL-RTL (MAC-2D con 3 Mult.)

XC4LX25-11 (%Uso)	1-Etapa	2-Etapa	4-Etapa	6-Etapa	24-Etapa
Slices	281(2 %)	573(5 %)	1149(10 %)	1719(15 %)	-
Flip-Flops	351(1 %)	721(3 %)	1442(6 %)	2163(10 %)	-
DSP48	8(16 %)	16(33 %)	32(66 %)	48(100 %)	-
BlockRAM	2(2 %)	4(5 %)	8(11 %)	12(16 %)	-
BUFGs	2(6 %)	2(6 %)	2(6 %)	2(6 %)	-
Nº ciclos CST/píxel	5	5	5	5	-
F. Max. CST (cir/DCM MHz)	436/315	348/315	348/315	337/315	-
F. Max. CLK (cir/DCM MHz)	72.6/52.5	58.0/52.5	58.0/52.5	56.1/52.5	-
GOPS (cir/DCM)	3.05/2.20	4.87/4.41	9.74/8.82	14.1/13.2	-

Carthagonova VHDL-RTL (MAC-2D con 9 Mult.)

XC4LX25-11 (%Uso)	1-Etapa	2-Etapa	4-Etapa	8-Etapa	24-Etapa
Slices	283(2 %)	564(5 %)	-	-	-
Flip-Flops	323(1 %)	640(2 %)	-	-	-
DSP48	23(47 %)	46(95 %)	-	-	-
BlockRAM	2(2 %)	4(5 %)	-	-	-
BUFGs	2(6 %)	2(6 %)	-	-	-
Nº ciclos CLK/píxel	1	1	-	-	-
F. Max. CLK (cir/DCM MHz)	362	361	-	-	-
GOPS (cir)	12.6	25.2	-	-	-

en posteriores versiones añadiendo un registro adicional a la salida de la unidad MAC y modificando la unidad de control de la Etapa. Este cambio tendría como objetivo centrar el procesamiento de la sub-etapa S2, entre los flancos del reloj CLK, para asegurar los tiempos de *setup* de las sub-etapas.

Las simulaciones temporales han determinado que la máxima frecuencia de funcionamiento de los circuitos es de 438 MHz, aunque las especificaciones de los DCMs indican que la máxima frecuencia sintetizable es de 315 MHz.

En cuanto al número máximo de Etapas que pueden ser implementadas en la FPGA, se ha comprobado que el límite viene determinado, nuevamente, por el número de DSP48 disponibles en el dispositivo. Hay que indicar que en las arquitecturas MAC-2D con más de un multiplicador el consumo de primitivas DSP48 es mayor al esperado, debido a que algunas de ellas han sido utilizadas como sumadores para incrementar la velocidad de procesamiento.

A partir de la frecuencia máxima de funcionamiento, y considerando imágenes de 640×480 píxeles codificadas con 8 bits, se deduce que la Etapa Carthagonova con un multiplicador por MAC procesará imágenes a $(438e6/11)/(640 \times 480) = 129$ fps, ofreciendo un ancho de banda de entrada de $2 \times (438/11) = 79.6$ MB/S. La arquitectura con tres multiplicadores procesará imágenes a $(436e6/5)/(640 \times 480) = 283$ fps, con un ancho de banda de $2 \times (436/5) = 174.4$ MB/S, mientras que la arquitectura con nueve multiplicadores procesará a $(362e6/1)/(640 \times 480) = 1178$ fps con un ancho de banda de entrada de $2 \times 362 = 724$ MB/S.

4.10. Comparación de los resultados de implementación de las arquitecturas Carthago y Carthagonova

En esta sección se comparan las diferentes implementaciones realizadas de las arquitecturas Carthago y Carthagonova, considerando para ello los resultados obtenidos en las secciones previas en cuanto a utilización de recursos hardware y velocidad de funcionamiento. Las figuras 4.32 y 4.33 muestran las gráficas que resumen los resultados obtenidos.

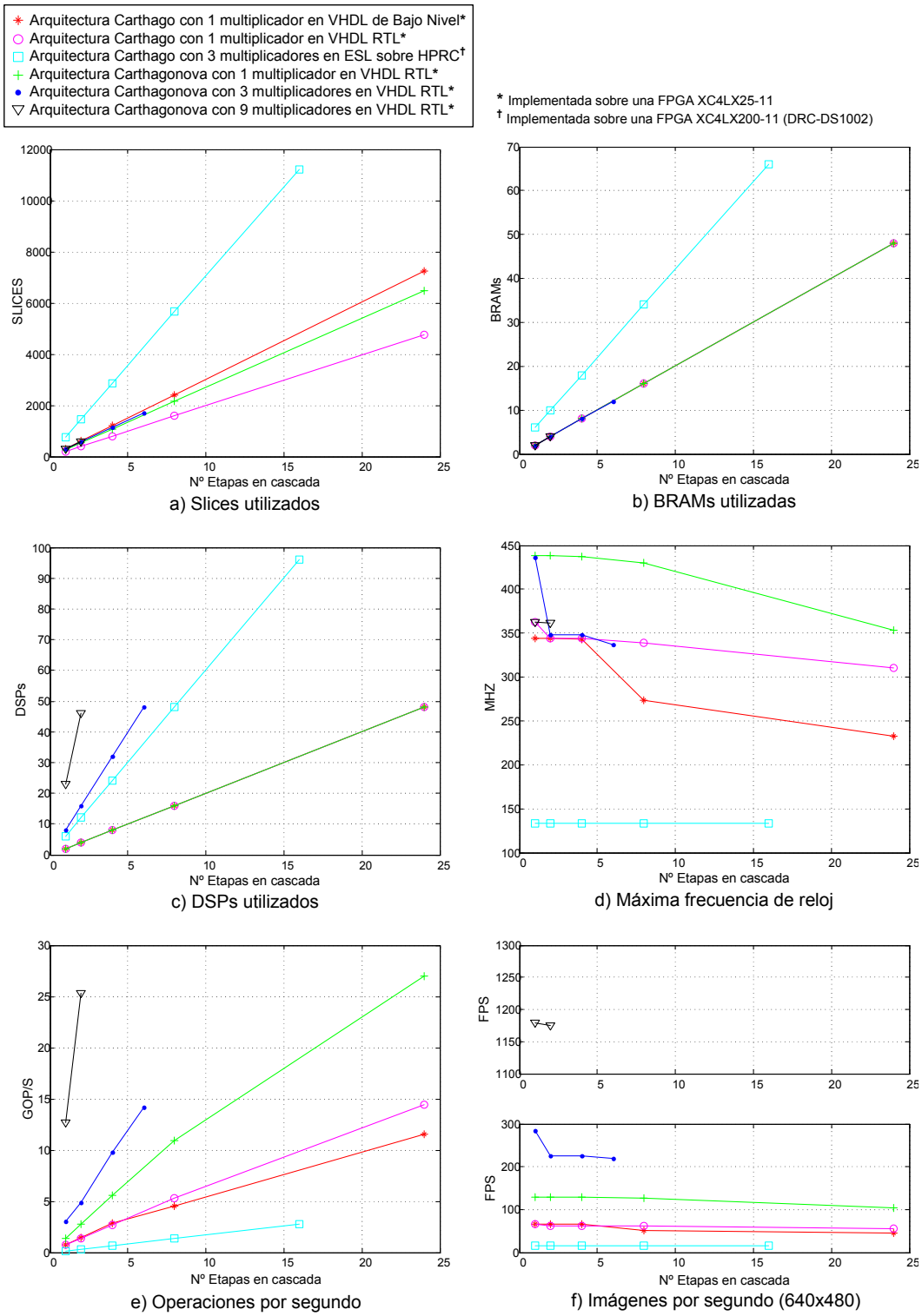


Figura 4.32: Comparación de los resultados de las implementaciones de las arquitecturas Carthago y Carthagonova.

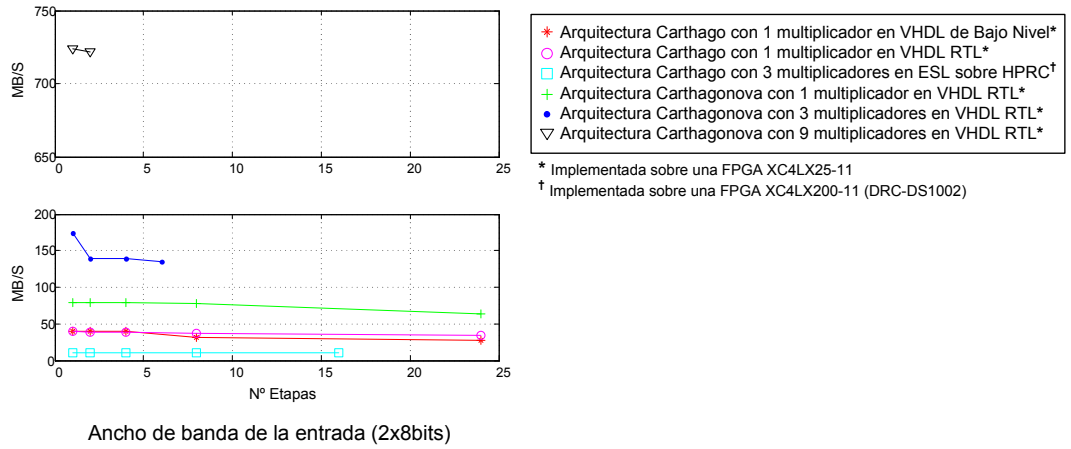


Figura 4.33: Comparación de los resultados de implementación de las arquitecturas Carthago y Carthagonova (continuación).

La figura 4.32-a muestra la relación entre el consumo de Slices y el número de Etapas utilizadas por cada una de las implementaciones. Como se observa, en todos los casos, el consumo de Slices crece proporcionalmente con el número de Etapas implementadas, destacando el mayor consumo de la realización en Impulse-C debido a su descripción algorítmica de mayor nivel, basada en ESL (*Electronic System Level*). Por otra parte, la arquitectura Carthago en VHDL-RTL es la que menos Slices utiliza ya que ha sido diseñada específicamente para minimizar los recursos lógicos de la FPGA. No obstante, la arquitectura Carthago implementada en VHDL de bajo nivel no obtiene los resultados esperados, debido a que el posicionamiento manual de los componentes no consigue aprovechar los recursos de la FPGA de forma tan eficiente como lo hace la herramienta de diseño automática.

La figura 4.32-b representa el consumo de bloques de memoria BRAMs en cada una de las implementaciones propuestas. Como era de esperar en todos los casos, la utilización de este recurso crece proporcionalmente con el número de Etapas. En las arquitecturas implementadas en VHDL, el consumo de BRAMs es el mismo en todos los casos, mientras que en la implementación en Impulse-C es comparativametne superior. Este consumo extra, por parte de la implementación en Impulse-C, se debe a la utilización de bloques de memoria adicionales para implementar los Buffer-stream de los procesos de comunicación.

La figura 4.32-c muestra la relación entre el consumo de primitivas DSP48 y el

número de Etapas implementación en cada caso. Como se observa, todas las arquitecturas con un multiplicador por unidad MAC utilizan la misma cantidad de DSP48, mientras que la arquitectura con nueve multiplicadores, lógicamente, es la que presenta mayor consume de este recurso. Hay que tener en cuenta, además, que esta arquitectura utiliza algunos DSP48 como sumadores para incrementar la velocidad de procesamiento de la Etapa. Entre las arquitecturas con tres multiplicadores por unidad MAC se comprueba que la arquitectura Carthagonova utiliza más recursos DSP48 que la implementación en Impulse-C, debido a que también utiliza algunas de estas primitivas como sumadores.

En cuanto a los resultados de velocidad, la figura 4.32-d muestra que la arquitectura Carthagonova funciona a mayor frecuencia que la arquitectura Carthago, principalmente debido a la eliminación del camino crítico entre las sub-etapas S1 y S2. En concreto, la arquitectura Carthagonova con un multiplicador por unidad MAC es la versión que consigue la máxima frecuencia de funcionamiento. La figura muestra además que las frecuencias de trabajo disminuyen con la complejidad de los circuitos y que, por esta razón, la arquitectura Carthagonova con tres multiplicadores por unidad MAC presenta un salto significativo entre las implementaciones de 1 y 2 Etapas. El escalón observado en la implementación de la arquitectura Carthago en VHD de bajo nivel se debe, fundamentalmente, a la rigidez estructural de la macro RPM (*Relationally Placed Macros*) de la Etapa y a la dificultad que encuentra la herramienta de síntesis para situar dichas macros cerca de los bloques BRAM y DSP48 de la FPGA. La frecuencia de funcionamiento de las implementaciones en Impulse-C es significativamente menor que las implementadas en VHDL e independiente del número de Etapas, ya que en todas los casos la RPU (*Reconfigurable Processor Unit*) fue configurada con la misma frecuencia de funcionamiento.

La figura 4.32-e muestra la capacidad de cómputo de las diferentes implementaciones expresada en GOPS (Giga Operaciones por Segundo), considerando 35 operaciones aritméticas por Etapa (18 multiplicaciones y 17 sumas). Como se observa en la figura, la arquitectura que presenta mayor capacidad de cómputo por Etapa es la Carthagonova con nueve multiplicadores por unidad MAC (12.6 GOPS), seguida por la arquitectura Carthagonova con tres multiplicadores por unidad MAC (3.05 GOPS) y la Carthagonova con un multiplicador por unidad MAC (1.39 GOPS). La implementación en Impulse-C consigue los peores resul-

tado de todos (0.17 GOPS) a pesar de utilizar tres multiplicadores por unidad MAC. En general, la arquitectura Carthagonova ofrece mejores prestaciones de cómputo que la arquitectura Carthago debido a su mayor frecuencia de funcionamiento y a la menor cantidad de ciclos de reloj CST que necesita para completar el procesamiento. En dicha figura se muestra también que el número de operaciones realizadas por las implementaciones crece linealmente con el número de Etapas y que, en general, la pendiente de esta curva es mayor cuanto mayor es el número de multiplicadores utilizados por las Etapas. Tras comparar las arquitecturas Carthago en VHDL-RTL y VHDL de bajo nivel se observa que ambas tienen una capacidad de cómputo similar, aunque a partir de un determinado número de Etapas la implementación a bajo nivel disminuye su rendimiento debido al peor comportamiento de la frecuencia de funcionamiento. Finalmente, es interesante destacar que la arquitectura Carthagonova con un multiplicador por unidad MAC obtiene las mejores prestaciones de cómputo para una ocupación completa de la FPGA (26.9 GOPS), seguida por la arquitectura con nueve multiplicadores por unidad MAC (25.2 GOPS).

La figura 4.32-f representa la cantidad de imágenes por segundo que pueden procesar cada una de las implementaciones, considerando imágenes de tamaño 640×480 codificadas con 8 bits de precisión. Como se aprecia en la figura, la Etapa Carthagonova con nueve multiplicadores por MAC destaca al alcanzar 1178 fps, le siguen la Etapa Carthagonova con tres multiplicadores por MAC con 283 fps, la Etapa Carthagonova con un multiplicador por MAC con 129 fps, la Etapa Carthago implementada a bajo nivel con 79 fps, la Etapa Carthago implementada en VHDL-RTL con 65 fps y, finalmente, como era de esperar, la Etapa Carthago implementada en Impulse-C, con 16 fps. Se observa que la velocidad de procesamiento decrece al aumentar el número de Etapas y que el descenso está directamente relacionado con la frecuencia de funcionamiento (figura 4.32-d). La figura 4.33 evidencia que el ancho de banda de la entrada y número de imágenes procesadas por segundo guardan una relación directa.

4.11. Análisis de los resultados

A partir de los resultados obtenidos en las secciones anteriores se ha podido comprobar que, en general, la arquitectura Carthagonova ofrece mejores prestaciones que la arquitectura Carthago. A continuación, se hará una valoración más específica de las diferentes versiones de la arquitectura Carthagonova.

Analizando los resultados de las implementaciones de la arquitectura Carthagnova, se puede comprobar que la arquitectura con nueve multiplicadores por unidad presenta mayor velocidad de procesamiento que las otras versiones a igual cantidad de Etapas. No obstante, dado que esta arquitectura tiene mayor consume de recursos de cómputo, con ella solamente es posible implementar 2 Etapas (en la FPGA seleccionada) frente a las 24 de la arquitectura con un multiplicador por MAC, lo que provoca que la capacidad de cómputo total de esta arquitectura (25.2 GOPS) sea inferior a la obtenida por la arquitectura con un multiplicador por MAC (26.9 GOPS). Las razones por las que esto ocurre se deben a que la arquitectura con un multiplicador funciona a mayor frecuencia, al ser más simple y estar super-segmentada, y a que presenta un factor de utilización de los DSP48 como multiplicadores del 100 % (2/2) frente al 78.2 % (18/23) de la arquitectura con nueve multiplicadores, lo que permite implementar mayor cantidad de Etapas. En el cuadro 4.4 se observa que la arquitectura con tres multiplicadores por MAC es la versión con menor capacidad de cómputo total (14.1 GOPS), debido, principalmente, al bajo factor de utilización de los DSP48 como multiplicadores, sólo del 75 % (6/8), y a la baja tasa de utilización de los multiplicadores por ciclo de reloj, sólo del 60 % (3/5) frente al 81.8 % (9/11) de la arquitectura con un multiplicador y al 100 % (1/1) de la arquitectura con nueve multiplicadores.

La gráfica de la figura 4.34 indica cuanto debería incrementarse la frecuencia de funcionamiento de las arquitecturas super-segmentadas, con respecto a lo no super-segmentada, para conseguir que n-Etapas super-segmentadas obtengan la misma capacidad de cómputo que una sola Etapa no super-segmentada, usando la misma cantidad de multiplicadores. La gráfica se ha obtenido considerando que la arquitectura no super-segmentada realiza las operaciones en un solo ciclo de reloj (CLK), mientras que las arquitecturas super-segmentadas utilizan $(2r+1)^2+1$ ciclos (CST), en el caso de un multiplicador por MAC, y $(2r+2)$ ciclos, en el caso de $(2r+1)$ multiplicadores por MAC, siendo r el radio de vecindad de la Etapa.

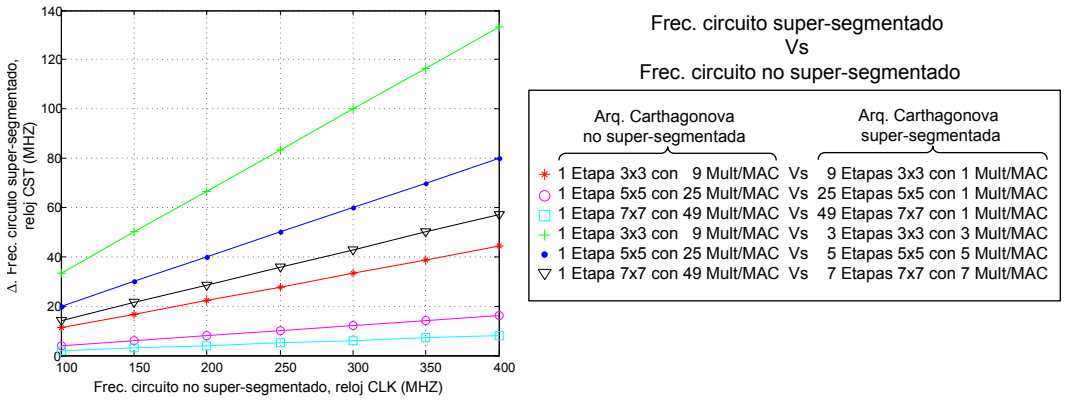


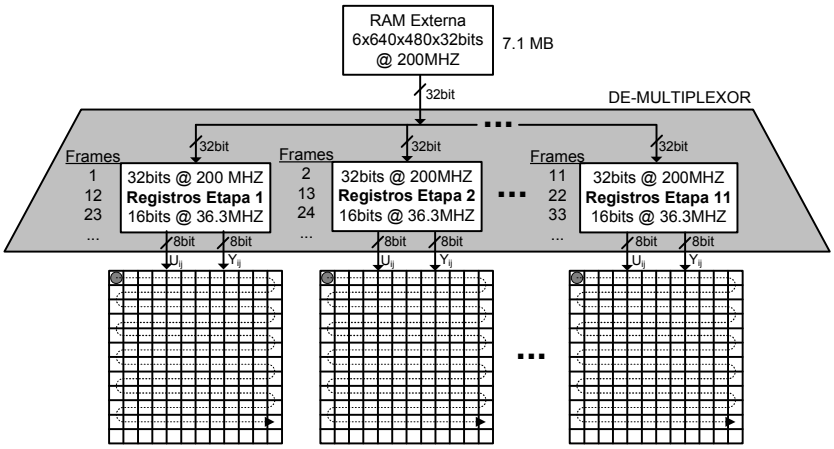
Figura 4.34: Incremento de la frecuencia del reloj CST para conseguir que n-Etapas superper-segmentadas consiga la misma capacidad de cómputo que una Etapa no super-segmentada usando la misma cantidad de multiplicadores.

A partir de la gráfica anterior, se deduce que para una misma capacidad de cómputo, las Etapas con un multiplicador por MAC requieren una frecuencia de funcionamiento menor que las Etapa con tres multiplicadores por MAC, puesto que realizan mayor número de multiplicaciones por ciclo de reloj. A esto hay que añadir que la arquitectura con un multiplicador puede funcionar a mayor frecuencia, al ser más sencillas, y que todos los DSP48 que utilizan trabajan como multiplicadores, lo que permitirá implementar mayor cantidad de Etapas.

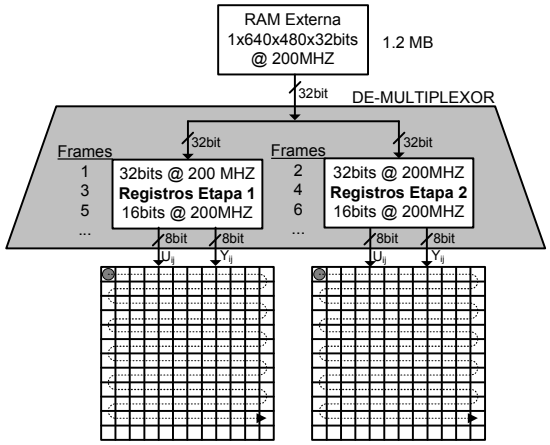
A partir de los resultados temporales, obtenidos tras implementación de las diferentes arquitecturas, se ha podido comprobar que las Etapas super-segmentadas, con un único multiplicador, tienen una frecuencia de funcionamiento media un 15.9 % superior a la obtenida por la Etapa no super-segmentada, dada su menor complejidad y la eliminación del camino crítico entre las sub-etapas S1 y S2.

Se ha podido corroborar también, que las Etapas Carthagonova tienen un ancho de banda de entrada que varía en función del número de multiplicadores utilizados por la arquitectura. Las pruebas realizadas, con datos de entrada de 8 bits, han demostrado que la Etapa con un multiplicador por MAC tiene un ancho de banda de 79.6 MB/S, la de tres multiplicadores por MAC de 174.4 MB/S y la de nueve multiplicadores por MAC de 724 MB/S. Hay que indicar, sin embargo, que el uso de varias Etapas en paralelo, junto con la utilización de memoria externa y una interfaz de adaptación, permitirá desarrollar aplicaciones con mayores anchos de banda.

La figura 4.35 muestra un ejemplo de dos sistemas equivalentes, constituidos por Etapas distintas, que han sido configurados para trabajar con el mismo ancho de banda de entrada (800 MB/S). El primer sistema está constituido por 11 Etapas con un multiplicador por MAC (figura 4.35-a), mientras que el segundo utiliza 2 Etapas con nueve multiplicadores por unidad (figura 4.35-b).



a) Interfaz de entrada para 11 Etapas paralelas con 1 mult./MAC



b) Interfaz de entrada para 2 Etapas paralelas con 9 mult./MAC

Figura 4.35: Sistemas equivalentes para un ancho de banda de entrada de 800 MB/S. a) Sistema compuesto por 11 Etapas paralelas con un multiplicador por unidad MAC. b) Sistema compuesto por 2 Etapas paralelas con nueve multiplicadores por unidad MAC.

Suponiendo imágenes de entrada de 640×480 píxeles y 8 bits de precisión, el sistema compuesto por Etapas con un multiplicador por MAC utilizará 11 Etapas y una memoria externa de 7.1 MB para almacenar y procesar 11 imágenes en

paralelo. El ancho de banda máximo alcanzado por este sistema será de $79.6 \times 11 = 875$ MB/S. El sistema formado por Etapas con nueve multiplicadores por unidad usará 2 Etapas y una memoria de 1.2 MB para almacenar y procesar 2 imágenes en paralelo. En este caso, el ancho de banda máximo del sistema será de $724 \times 2 = 1448$ MB/S. Las principales ventajas del sistema compuesto por Etapas con un multiplicador por MAC serán las siguientes: menor utilización de multiplicadores, 11 frente a los 18 usados por el segundo sistema (mayor diferencia si comparamos el uso de DSP48), mayor adaptabilidad a variaciones en el ancho de banda, dado que la eliminación o incorporación de nuevas Etapas permitirá un ajuste más fino de este parámetro, y la utilización de señales de reloj externa de menor frecuencia, 36.6 MHZ frente a los 200 MHZ del segundo sistema. Por contra, los principales inconvenientes del primer sistema serán el empleo de una memoria externa de mayor capacidad y el uso de una interfaz de adaptación del ancho de banda más compleja.

La arquitectura Carthagonova ha sido diseñada para trabajar con imágenes de vídeo en tiempo real sin necesidad de utilizar memorias externas, la figura 4.36 representa el esquema propuesto para ello. En dicho esquema, las imágenes de entrada son procesadas en flujo de datos, conforme son generadas por la fuente de video. Por otra parte, las imágenes resultantes podrán utilizar las mismas señales de sincronismo que las imágenes de entrada, siempre y cuando la latencia introducida en el procesamiento sea múltiplo del tiempo empleado para generar las imagenes. En caso contrario, las señales de sincronización deberán retrasarse el mismo tiempo que el empleado en el procesamiento.

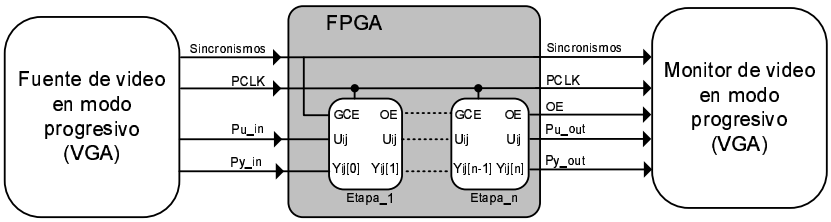


Figura 4.36: Sistema de procesamiento de video en tiempo real basado en la arquitectura Carthagonova.

A partir de la latencia de una Etapa es posible determinada el número de Etapas que habrá que conectar en cascada para conseguir un retraso equivalente al empleado en la generación de una imágenes. Evidentemente, si el ancho de banda

de la entrada lo permite, lo ideal sería emplear Etapas con un multiplicador por unidad MAC para aprovechar mejor los recursos internos de la FPGA. El punto marcado en la figura 4.37 representa el número de Etapas con un multiplicador por MAC que se han de conectar en cascada para genera un retraso equivalente a 640×480 píxeles.

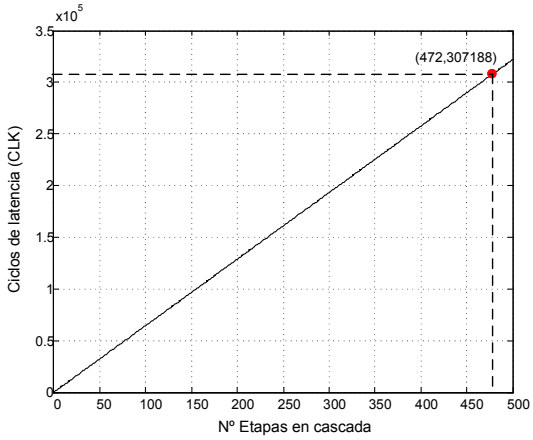


Figura 4.37: Latencia de la arquitectura Carthagonova en función del número de Etapas conectadas en cascada, considerando imágenes de entrada 640×480 píxles y Etapas de radio de vecindad 1 y un multiplicador por unidad MAC.

4.12. Prestaciones de la arquitectura Carthagonova sobre Virtex-6 y Virtex-7

La Etapa Carthagonova con un multiplicador por unidad MAC, particularizada para entradas de datos de 9 bits, ha sido implementada sobre las familias de FPGAs Virtex-6 y Virtex-7 de Xilinx. El dispositivo Virtex-6 seleccionado (XC6VSX475t-2) cuenta con un grado de velocidad medio y la máxima cantidad de primitivas BRAMs y DSP48s disponibles de toda la familia (2128 BRAMs y 2016 DSP48). Por su parte, la FPGA Virtex-7 utilizada (XC7VX485t-3) tiene el máximo grado de velocidad y un número de primitivas BRAM y DSP48s que la sitúan en la media de esta familia (2360 BRAMs y 2880 DSP48). El cuadro 4.5 resume los resultados de la implementación de 1 y 800 Etapas sobre dichas FPGAs, usando la herramienta de síntesis XST de Xilinx ISE 13.1.

Cuadro 4.5: Resultados de la implementación de la Etapa Carthagonova con un multiplicador por MAC en VHDL a nivel RTL sobre las FPGAs XC6VSX475t-2 y XC7VX485t-3 de Xilinx.

Carthagonova (MAC-2D con 1 Mult.)

	XC6VSX475t-2		XC7VX485t-3	
(%Uso)	1-Etapa	800-Etapa	1-Etapa	800-Etapa
Slices	76(<1 %)	71944(96 %)	94(<1 %)	74046(97 %)
Flip-Flops	330(<1 %)	288800(52 %)	330(<1 %)	288800(51 %)
DSP48	2(<1 %)	1600(79 %)	2(<1 %)	1600(57 %)
BlockRAM	2(<1 %)	1600(75 %)	2(<1 %)	1600(77 %)
BUFGs	2(6 %)	3(9 %)	2(6 %)	3(9 %)
Nº ciclos CST/píxel	11	11	11	11
F. Max. CST (cir MHz)	550	295	557	393
F. Max. CLK (cir MHz)	50	26	50	35
Opera./s (cir GOPS)	1.75	728	1.75	980

Los resultados de la simulación temporal han determinado que la máxima frecuencia de funcionamiento de una Etapa es de 550 MHz para la Virtex-6 y de 557 MHz para el caso de la Virtex-7. Para el circuito formado por 800 Etapas, las frecuencias de funcionamiento máximas alcanzadas fueron de 259 MHz y 393 MHz, respectivamente. Hay que indicar que en estas dos familias, las primitivsa utilizadas para sintetizar la frecuencia del reloj CST, los MMCM (Mixed Mode Clock Managers), pueden trabajar con frecuencias superiores a 700 MHz.

A partir de la frecuencia de funcionamiento máxima es posible deducir que la Etapa implementada sobre la Virtex-6 podrá procesar imágenes de 640×480 píxeles a $(550e6/11)/(640 \times 480) = 162$ fps, mientras que el circuito formado por las 800 Etapas lo hará a $(295e6/11)/(640 \times 480) = 87$ fps. En cuando al número de operaciones realizadas por segundo (considerando 35 operaciones por Etapa), una Etapa en la Virtex-6 realizará 1.75 GOPS, mientras que 800 Etapas realizarán 728 GOPS. Para el caso de la Virtex-7, el número de imágenes procesadas por una Etapa será de $(557e6/11)/(640 \times 480) = 164$ fps y en el caso de 800 Etapas de $(393e6/11)/(640 \times 480) = 116$ fps. Finalmente, el número de operaciones por segundo ejecutadas por una Etapa en la Virtex-7 será 1.75 GOPS, mientras que el circuito de 800 Etapas ejecutará 980 GOPS.

En lo que respecta al consumo de área, se ha comprobado que ni las primitivas DSP48s ni las BRAMs son los recursos críticos que limitan la implementación de la arquitectura sobre estas FPGAs. En concreto, los recursos más utilizados han sido los Slices, siendo necesario ocupar, en la implementación del circuito de 800 Etapas, el 96 % en la Virtex-6 y el 97 % en la Virtex-7. Hay que indicar que los Slices de estas familias presentan una organización interna diferente a los Slices de la Virtex-4, ya que contienen el doble de LUTs y una cantidad cuatro veces superior de Flip-Flops. Considerando que el circuito con 800 Etapas está organizado como una estructura neuronal formada por 800 Capas de una sola Celda y una sola iteración, el sistema permite emular, aproximadamente, el equivalente a 245.7 millones de celdas CNN tradicionales, procesando imágenes de video de 640×480 píxeles en tiempo real (116 fps en el caso de la Virtex-7).

Aunque no ha sido posible implementar los circuitos sobre una Virtex-7 diferente, ya que requieren licencias específica, las estimaciones realizadas en cuanto al número máximo de Etapas que pueden ser implementadas en la FPGA de mayor tamaño de esta familia (XC7VX1140t) son de 1680 Etapas, limitados en este caso por el número de DSP48s disponibles (3360).

4.13. Características principales de la arquitectura Carthagonova

La arquitectura Carthagonova ha sido diseñada para emular el procesamiento realizado por las redes CNN en tiempo real. Uno de los principales objetivo ha sido desarrollar una arquitectura versátil que pudiera ser fácilmente adaptable al mayor número posible de aplicaciones, evitando, además, la estructura rígida impuesta por la implementación tradicional de la red CNN. En los apartados siguientes se resumen las principales características de la arquitectura Carthagonova, desde el punto de vista de los fundamentos que se han considerado para su desarrollo: el modelo de la red CNN, la arquitectura externa, la modularidad del sistema y la arquitectura interna.

Modelo de la red CNN

El proceso de desarrollo de la arquitectura Carthagonova parte de la aproximación de Euler del modelo FSR de la red CNN estándar. Para particularizar dicho modelo se han tenido en cuenta las siguientes consideraciones: condiciones de contorno Dirichlet, plantillas variantes, tanto en el espacio como en el tiempo, y entrada de datos U_{ij} que puede influir en la salida del modelo en sucesivas iteraciones. Esta última característica puede ser de interés para desarrollar aplicaciones de procesamiento que requieran el uso de componentes del tipo "interneurona", es decir, aplicaciones donde interesa que la entrada del sistema pueda ser propagada y procesada por distintos elementos del sistema simultáneamente. El modelo de la red CNN propuesto ha sido estructurado en diferentes niveles de abstracción para facilitar su realización sobre diferentes plataformas de desarrollo y el uso de diferentes técnicas y lenguajes de implementación. El algoritmo obtenido, a partir del modelo propuesto, ha sido adaptado e implementado sobre diferentes plataformas: sobre FPGAs, usando VHDL a diferentes niveles, sobre un PC convencional, usando Matlab, y sobre un HPRC (High *Performance Reconfigurable Computing*) usando ANSI C estándar e Impulse-C.

El procedimiento utilizado para trasladar el modelo propuesto a las diferentes plataformas de desarrollo se ha centrado en encontrar de una solución sencilla y eficiente desde el punto de vista del uso de los recursos de cómputo y de la velocidad de procesamiento. Las distintas fases utilizadas en este procedimiento han dado lugar a un esquema de procesamiento en flujo de datos, constituido por un proceso recurrente, que ha sido desenrollado en un serie de Etapas en cascada para incrementar el paralelismo del sistema y la velocidad de procesamiento. Este esquema ha ayudado a definir una arquitectura externa que permite el diseño de sistemas CNN complejos con un coste en recursos hardware relativamente bajo. La arquitectura externa se basa en un conjunto de componentes y una interfaz de entrada/salida sencillos que facilitan el diseño y el desarrollo de las aplicaciones.

Arquitectura externa

Las principales ventajas de la arquitectura externa surgen de la flexibilidad y versatilidad de la estructura jerárquica del sistema, así como de la posibilidad de que su descripción hardware pueda ser descrita utilizando diversas técnicas y

herramientas de diseño, como: VHDL, System Generator, Impulse-C, etc.

Cada componente de la jerarquía opera en paralelo y ofrece la suficiente flexibilidad y modularidad para facilitar, de manera natural, la ejecución simultanea de varias operaciones de procesamiento. Esta organización jerárquica facilita la abstracción del procesamiento en diferentes estratos que manipulan la información, cada vez, a mayor nivel, realizando operaciones más complejas y generales. Los componentes de la jerarquía permiten simplificar los detalles de la realización hardware de los niveles inferiores y aumentar la modularidad del sistema, lo que facilitará su adaptación a los requisitos de las aplicaciones. Una característica importante de los componentes de nivel superior es que pueden ser diseñados, implementados y probados de forma independiente, por lo que es posible añadir nuevas funcionalidades al sistema sin necesidad de modificar los componentes ya definidos. Esto facilita el diseño y la adaptación de los componentes a las características del sistema y los requerimientos de las nuevas aplicaciones.

Modularidad del sistema

La flexibilidad y sencillez de la arquitectura externa se ponen de manifiesto con las facilidades que ofrece para el desarrollo de sistemas complejos formados por un conjunto indefinido de módulos de procesamiento interconectados. Estos módulos deben definir y respetar la interfaz de la arquitectura, para poder ser incorporados al sistema. La interfaz se encarga de recoger los datos y las señales de sincronismo generados por los módulos anteriores, y de enviar los resultados y nuevos sincronismos a los siguientes módulos de la cadena. El que todos los módulos utilicen la misma interfaz de comunicación facilita la interacción de los componentes y la incorporación de nuevos módulos al sistema. La escalabilidad, fomentada por la modularidad y la independencia de los componentes de la arquitectura, permite aprovechar las capacidades del sistema para añadir nuevas funcionalidades de procesamiento con sólo incluir nuevos módulos a la cadena. La utilización de módulos, con la capacidad suficiente para implementar el número adecuado de Etapas, permitirá que el sistema opere en tiempo real sin necesidad de utilizar memorias externas.

Arquitectura interna

La arquitectura interna de la Etapa Carthagonova ha sido diseñada para procesar la información en tiempo real, aprovechando los recursos internos (de cálculo y memoria) de las FPGAs. Los requisitos temporales y de área exigidos a la arquitectura han sido prioritarios en el diseño y han servido para analizar el rendimiento de las diferentes versiones de la Etapa propuestas. Para lograr un compromiso adecuado entre área y velocidad se ha optado por desarrollar una arquitectura super-segmentada basada en dos regiones de reloj: una con menor frecuencia de funcionamiento, encargada de gestionar la interfaz de entrada/salida de la Etapa, y otra con mayor frecuencia, encargada de realizar el procesamiento intensivo de los datos. La principal ventaja de esta solución reside en la posibilidad de dividir el circuito de la Etapa en dos partes independientes que pueden ser tratadas y optimizadas por separado para aumentar el rendimiento global del conjunto.

En cuanto al consumo de área del circuito, la utilización de dos regiones de reloj ha permitido optimizar el uso de los recursos de cálculo y maximizar el número de Etapas que pueden ser implementadas en los dispositivos. Las unidades de procesamiento de la arquitectura han sido diseñadas para trabajar con: 1, $(r+1)$ y $(r+1)^2$ multiplicadores por unidad MAC con el fin de analizar el rendimiento de diferentes versiones de la Etapa. En este sentido, se ha podido comprobar que las Etapas con un multiplicador por unidad MAC alcanzan, proporcionalmente, un rendimiento computacional superior a las arquitecturas con más multiplicadores. La Etapa ha sido diseñada para realizar las operaciones en punto fijo a máxima precisión, es excepto en el caso de la variable de salida que ha sido truncada a la precisión de la entrada.

Un aspecto a destacar en el diseño de la arquitectura de la Etapa ha sido el aprovechamiento de los recursos de memoria internos de la FPGA. La Etapa Carthagonova ha sido desarrollada para almacenar la mínima cantidad de información requerida por el procesamiento, sin necesidad de utilizar memoria externa. La utilización de memoria interna, junto al uso eficiente de los recursos de cálculo, permite desarrollar sistemas embebidos simples que pueden trabaja a alta velocidad. Otra ventaja, es que las Etapas almacenan en su interior los diferentes juegos de plantillas y no utiliza la interfaz de entrada y salida para seleccionar la plantilla, lo que disminuiría el ancho de banda de la entrada.

Capítulo 5

Aplicación de la arquitectura CNN y sistema de desarrollo

En este capítulo, la arquitectura CNN propuesta ha sido aplicada al diseño de un modelo artificial, relativamente simple, de la primera sinapsis de la retina del ojo. El modelo considera los principales circuitos neuronales de la retina, encargados del procesamiento espacial, y tiene como objetivo analizar como la percepción del contraste y la resolución visual pueden verse afectadas por factores tales como la convergencia de la información o la inhibición lateral. El modelo artificial de la retina se ha desarrollado para ser emulado con la arquitectura CNN propuesta y llevar a cabo sus tareas de procesamiento en tiempo real.

En este capítulo se describe también el diseño y la implementación de un sistema embebido múlti-FPGA, destinado a desarrollar aplicaciones de procesamiento de vídeo en tiempo real, que requieran el uso de redes CNN complejas y de gran tamaño. El sistema dispone de una entrada de vídeo en modo progresivo y de una salida de vídeo compatible con el estándar VGA. Las principales características del sistema vienen determinadas por la modularidad y flexibilidad de la arquitectura CNN, la cual facilita la expansión del sistema y su adaptación a aplicaciones de diferente tamaño. El sistema está formado por un número indefinido de módulos de procesamiento, cada uno de ellos consistente en una tarjeta FPGA, entre los que se distribuye la estructura de la red CNN. Los módulos están interconectados entre sí a través de una sencilla interfaz de comunicación común,

que facilita la ampliación del sistema. La capacidad de cómputo del sistema vendrá determinada por el número total de módulos utilizados y por la cantidad de recursos disponibles en las FPGAs. Uno de los principales objetivos considerados en el diseño ha sido que la capacidad cómputo del sistema pudiera ser ampliada de forma rápida y sencilla. El sistema incluye también un módulo framegrabber que se encarga de adaptar la información de vídeo procesada a los sincronismos de vídeo de la salida. La modularidad y ampliabilidad de esta plataforma hardware hace que sea un excelente banco de pruebas para desarrollar, de forma rápida y sencilla, aplicaciones de procesamiento en tiempo real basadas en redes CNN.

5.1. Aplicación de la arquitectura CNN al diseño de un modelo artificial de la primera sinapsis de la retina

5.1.1. Introducción a la morfología interna de la retina

La retina (Kolb et al., 2011) es un fino tejido del ojo, formado por células sensibles a la luz, clave en el sistema visual primario y considerada, junto con el cerebro y la médula espinal, parte del sistema nervioso central. La retina está compuesta por una serie de capas neuronales con estructura similar pero con diferente funcionalidad y morfología. Cada parte de la retina realiza un proceso diferente, en función de los estímulos lumínicos que recibe, con el fin de extraer las características de la información visual que se percibe. La figura 5.1 muestra en esquema de morfología de la retina.

A pesar de que la retina es un fino tejido, de un grosor aproximado de 0.2-0.5 mm, está formada por 10 capas diferentes y contiene al menos 6 tipos básicos de células neuronales (Masland 2001). Los cuerpos de las células se distribuyen sobre la retina a lo largo de tres capas: la capa nuclear externa, donde se localizan los fotorreceptores; la capa nuclear interna, que contiene los cuerpos de las células bipolares, horizontales, amacrinas e interplexiformes; y la capa de células ganglionales, donde se localizan los cuerpos de las células ganglionales. Las conexiones sinápticas entre las células de la retina se extienden principalmente a lo largo de

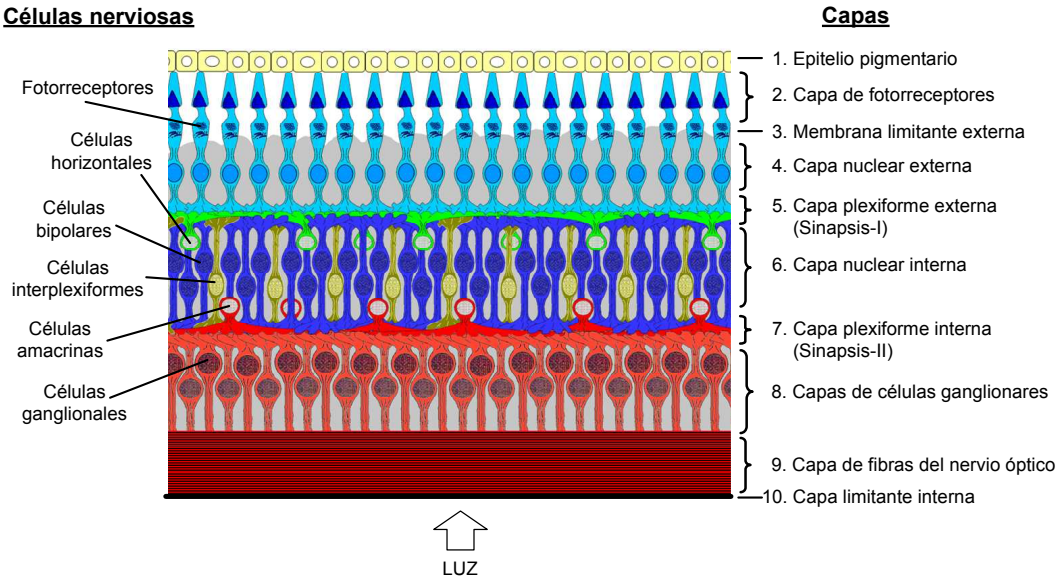


Figura 5.1: Esquema de la sección transversal de la región central de la retina.

dos capas: la capa plexiforme externa (sinapsis-I), consistente en una densa red de sinápsis entre los fotorreceptores y las células de la capa nuclear interna; y la capa plexiforme interna (sinapsis-II), donde se entrelazan las conexiones de las células de la capa interna y de las células ganglionares. En la sinapsis-I se lleva a cabo el principal procesamiento espacial de la retina, mientras que en la sinapsis-II se realiza el procesamiento temporal de la información visual.

Una de las principales funciones de la retina es transformar la luz en impulsos nerviosos. Esta traducción se inicia en las células fotorreceptoras: conos y bastones, y supone la modificación de las propiedades electro/químicas de estas células en respuesta a los estímulos lumínicos que reciben. La señales generadas por los fotorreceptores son procesadas por las células internas de la retina y propagadas hacia las células ganglionares a través de la sinápsis. Las células ganglionales finalmente, son las responsables de codificar las diferentes características visuales detectadas en la retina y de transmitir dicho pre-procesamiento a los centros superiores del sistema visual (núcleo geniculado lateral y corteza visual), donde los estímulos sensoriales son definitivamente percibidos.

Las células nerviosas de la retina (Urtubia 2006; Willian et al., 1992): células bipolares, horizontales, amacrinas y ganglionares, se conectan entre sí para formar diferentes circuitos neuronales. Estos circuitos son los encargados de extraer las

diferentes características visuales de la información recibida por la retina.

Los circuitos neuronales de la retina presentan complejidades diferentes que varían desde esquemas muy simples, formados por unas pocas neuronas y conexiones sinápticas aferentes, hasta formaciones altamente complejas, compuestas por cientos de miles de células recurrentes y con altas densidades de conexiones, tanto aferentes como eferentes. En la mayoría de los circuitos de la retina, las señales proporcionadas por las células fotoreceptoras son transmitidas a una o unas pocas células neuronales. Estas neuronas, a su vez, integran la información proveniente de otras neuronas de alrededor, constituyendo los circuitos que detectan las características visuales de la información recibida. Las células horizontales y amacrinas interactúan lateralmente a lo largo de las vías aferentes de la retina, formada por los fotoreceptores, las células bipolares y las ganglionares.

La complejidad de los circuitos neuronales de la retina depende de sus características fisiológicas, psicofísicas y neuroanatómicas, y está relacionada, de alguna forma, con la posición que ocupan los circuitos en la retina. La retina puede ser dividida en varias zonas en función de la proyección de su campo visual. En la zona central de la retina se localiza la fóvea, la cual contiene la mayor densidad de conos de toda la retina. En esta región, los conos están asociados a una sola célula ganglionar, lo que implica un ratio de convergencia aproximado de 1:1 (Drasdo et al., 2007). Además, las interacciones laterales entre células son las más bajas (Kolb et al., 2011), lo que en conjunto ocasiona que sea la región de la retina con la mayor resolución visual de todas. La región central incluye también la zona macular donde, conforme nos alejamos del centro, la densidad de conos comienza a disminuir y la de bastones a crecer. La mácula está cubierta por un pigmento amarillento que afecta a la percepción del color y que tiene como misión proteger a esta región de radiaciones luminosas de onda corta. En las regiones periféricas de la retina, la densidad de conos continúa disminuyendo y aumentando la de bastones. En estas zonas las interconexiones laterales abarcan cada vez mayor superficie, lo que provoca que las imágenes se vuelvan cada vez más borrosas e imprecisas. En las regiones más externas de la retina, las células ganglionares suelen estar asociadas a cientos de fotorreceptores, estableciéndose convergencias superiores a 200:1 (Urtubia 2006). Dicha convergencia e interacción lateral explican la baja agudeza visual de la región externa de la retina, la cual es aproximadamente del 2.5 % de la obtenida en la fóvea (Willian et al., 1992).

La retina puede ser dividida en las siguientes regiones: fóvea, foveola, parafóvea, perifóvea y zona periférica próxima, periférica media, periférica lejana y periférica externa. Estas regiones cuentan con circuitos neuronales con estructura regular, baja interacción lateral y convergencia limitada, lo cual recuerda a la topología y al conexionado de las redes CNN, que procesan la información localmente y únicamente la transmiten entre las células vecinas. Teniendo en cuenta estas similitudes, a continuación se propone el desarrollo de un modelo artificial de la primera sinapsis de la retina, en el que se consideran las principales características de sus circuitos neuronales. Dicho modelo ha sido planteado para ser implementado mediante la arquitectura CNN propuesta, aprovechando sus características y capacidad de procesamiento en tiempo real.

En los últimos años han salido a la luz diferentes modelos artificiales de la retina, unos basados en filtros no lineales (Herault 1996) y otros en redes CNN (Balya et al., 2002; Nagy et al., 2005). No obstante, aunque estos modelos han sido diseñados para ser proyectados sobre hardware, en realidad, debido a su alta complejidad y a las limitaciones de la tecnología, dichos modelos únicamente han podido ser simulados. En concreto, uno de los principales problemas en la realización hardware de los modelos de la retina, es llevar a cabo a implementación de la gran cantidad neuronas que son requeridas. En la actualidad, la mayoría de los sistemas hardware de desarrollo basados en redes CNN, cuentan con la posibilidad de implementar unos pocos miles de células por capa, mientras que solamente en la región de la fóvea existen aproximadamente unos 200.000 conos y unas 200.000 células ganglionares, conectadas a través de varias capas formadas por cientos de miles de neuronas que interactúan entre sí (Kolb et al., 2011). El modelo neuromórfico de la retina que se propone a continuación ha sido diseñado para utilizar la arquitectura CNN Carthagonova y emular en tiempo real parte del procesamiento espacial que realiza la primera sinapsis de la retina.

5.1.2. Modelo de la sinapsis-I de la retina basado en campos receptores

La organización interna de la primera sinapsis de la retina presupone que su principal cometido es llevar a cabo el procesamiento espacial de la información (Urtubia 2006). En ella, las conexiones sinápticas y la cantidad de fotorrecep-

tores que influyen sobre las neuronas de la capa interna son clave para determinar la resolución espacial y las características visuales de la información recibida.

La aproximación al modelo biológico de la retina propuesta está basada en el principio de sumación espacial de los campos receptores de la retina (Kolb 2003), es decir, en la suma algebraica de los diferentes estímulos recibidos en un circuito neuronal. El grado de convergencia de estos estímulos, su distribución, su intensidad y la inhibición lateral del circuito determinarán la disposición de las neuronas para el disparo.

El campo receptor de una célula (Dacey et al., 2000) se define como la región de la retina cuya estimulación lumínica desencadena el disparo o la inhibición de una célula. Como se muestra la figura 5.2, el campo receptor de una célula retiniana está formado por una zona central y otra periférica; el centro representa a los fotorreceptores que influyen de forma directa sobre la célula, mientras que la periferia simboliza a los fotorreceptores que influyen de forma indirecta, a través de las interconexiones laterales (Packer y Dacey 2002). En el modelo artificial propuesto se ha tenido en cuenta que las células bipolares son estimuladas directamente por los fotorreceptores y transversalmente por las células horizontales, las cuales, a su vez, son excitadas directamente por los fotorreceptores (Boycott et al., 1987). El tamaño del campo receptor de las células de la retina dependerá fundamentalmente de la superficie cubierta por las interconexiones laterales de las neuronas transversales (Drasdo et al., 2007).

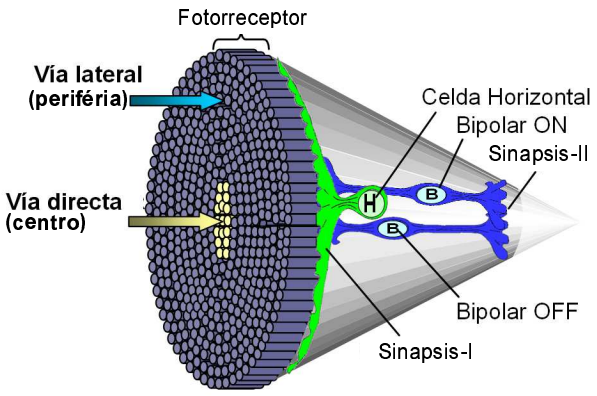


Figura 5.2: Campo receptor de las células bipolares y cono de convergencia.

En la fóvea, los conos suelen estar conectados a dos células bipolares llamadas bipolar ON y bipolar OFF. Estas células modulan su respuesta en función de

la intensidad de los estímulos que reciben, y presenta campos receptores opuestos. En las células tipo ON, un aumento de la intensidad luminosa en la zona central provocará un incremento en la intensidad del disparo, mientras que en la célula OFF lo disminuirá. Como se muestra en la figura 5.3, la respuesta del campo receptor de ambas células coincide con un volumen en forma de sombrero mejicano, cuyo centro y periferia responden de forma opuesta ante los mismos estímulos luminosos. Para el caso del campo receptor ON, la estimulación positiva del centro aumentará la activación de la célula, sin embargo, ese mismo estímulo en la periferia la disminuirá.

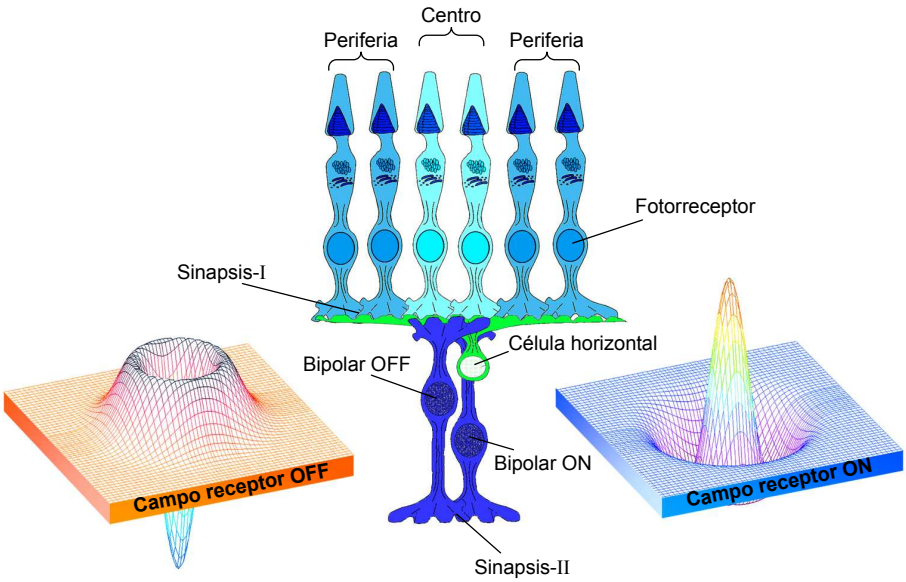


Figura 5.3: Modelo de la sinapsis-I de la retina y respuesta de los campos recpetores ON y OFF.

La curva de distribución de las respuestas de los campos receptores deriva de la superposición de dos curvas gaussianas con distinto signo: una excitadora (positiva) y otra inhibidora (negativa). Para el caso del campo receptor ON, la gaussiana excitadora representa la respuesta de la célula bipolar a estímulos directos de los fotorreceptores, mientras que la gaussiana inhibidora representa la respuesta de las células horizontales. La influencia lateral de las células horizontales en las células bipolares implicará la superposición de los dos efectos y justifica la forma de sombrero mejicano de la respuesta del campo receptor.

5.1.3. Aproximación a la sinapsis-I de la fóvea basada en redes CNN no lineales

El principio de sumación espacial observado en los campos receptores de las células de la retina ha sido adoptado para desarrollar un modelo artificial de la primera sinapsis de la fóvea, basado en redes CNN no lineales.

El modelo considera que 25 fotoreceptores (Boycott et al., 1987) influyen lateralmente, a través de una célula horizontal, sobre dos célula bipolares (una ON y otra OFF) conectadas directamente a un solo fotoreceptor. Bajo esta configuracion, la célula horizontal del circuito ha sido asociada a una matriz cuadrada de 5×5 fotorreceptores y las células bipolares al elemento central de dicha matriz. Por otra lado, tal y como se dijo en la sección 4.6.2, dado que n-Etapas CNN en cascada (de radio $r = 1$) implica una salida dependiente de $(2rn+1)^2$ entradas, la utilización de 2 Etapas será suficiente para implementar la Celda horizontal equivalente, figura 5.4. Las celdas bipolares se implementarán con una Etapa CNN.

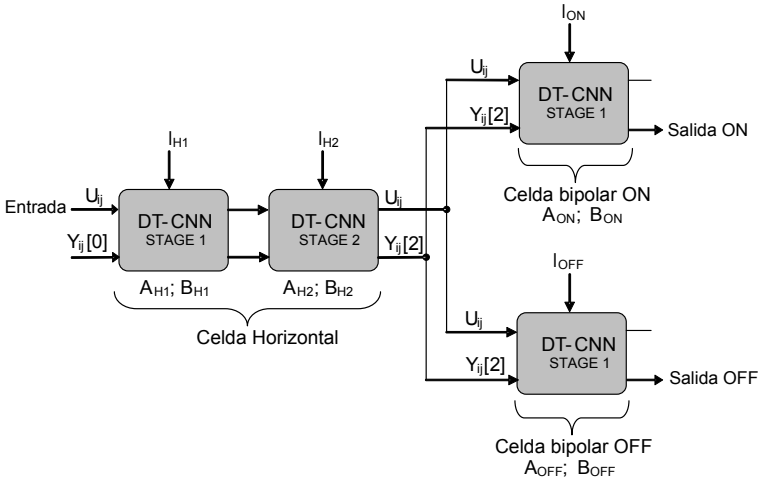


Figura 5.4: Modelo de la sinapsis-I de la fóvea basado en la arquitectura CNN.

Las entradas de la Celda horizontal deben influir por igual en su respuesta, sin llegar a saturarla. Para ello, todos los elementos del operador B_{H1} de la primera Etapa han sido normalizados e igualados. Además, puesto que esta primera Etapa no requiere la influencia de celdas vecinas, el operador A_{H1} de la plantilla ha sido igualado a cero. El operador de bias I_{H1} de esta Etapa también se ha establecido

a cero para no alterar el nivel de brillo de la salida. En la ecuación 5.1 se muestran la plantilla de la primera Etapa de la Celda horizontal.

$$A_{H1} = 0, B_{H1} = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}, I_{H1} = 0 \tag{5.1}$$

La salida de la siguiente Etapa de la Celda horizontal debe depender únicamente de la salida Y de la etapa anterior, la salida U solamente será propagada hacia la salida de la Celda. En esta segunda Etapa, el operador B_{H2} de la platilla ha sido igualado a cero, mientras que los pesos del operador A_{H2} han sido configurados con un patrón de distribución normal, calculado a partir de la ecuación 5.2.

$$G[i,j] = \frac{1}{2\pi\sigma^2}\, e^{-\frac{i^2+j^2}{2\sigma^2}} \tag{5.2}$$

Donde σ ha sido fijado experimentalmente a un valor de 0.7 y las variables i y j representan, respectivamente, los indices fila y columna de los elementos del operador con respecto al centro de la matriz. El bias I_{H2} de la segunda Etapa ha fijado a cero por la misma razón que en la primera Etapa. La ecuación 5.3 muestra la configuración de la plantilla de la segunda Etapa de la Celda horizontal.

$$A_{H2} = G_{3\times3}, B_{H2} = 0, I_{H2} = 0 \tag{5.3}$$

En cuanto a las células bipolares, dado que cada una de ellas recibe la información de un fotoreceptor y de la célula horizontal, el modelo hardware propuesto considera que cada célula bipolar estará representada por una Celda CNN de una sola Etapa, como muestra la figura 5.4. El puerto de entrada U_{ij} de las dos Celdas bipolares ha sido conectado a la salida de la Celda horizontal que propaga el flujo de datos de la entrada, el cual representa el estímulo directo de los fotorreceptores. Los elementos del operador B, de las Celdas bipolares, han sido igualados a cero, excepto el elemento central que representa el peso de la conexión directa entre el fotorreceptor y la Celda bipolar.

Para el caso de la Celda bipolar ON, el valor del elemento central del operador B es positivo, mientras que para la Celda bipolar OFF es negativo. El valor de dichos elementos determina la intensidad de la respuesta de la región central del

campo receptor de las Celdas bipolares. Los valores han sido elegidos experimentalmente (ecuación 5.4) con el fin de proporcionar nitidez a la imagen y un nivel de contraste adecuado para visualizar la respuesta del modelo.

Por otro lado, la salida Y_{ij} de la Celda horizontal ha sido conectada a la entrada Y_{ij} de las Celdas bipolares para poder llevar a cabo la sumación del efecto lateral. Dado que los estímulos periféricos de las Celdas bipolares deben ser contrarios a los del centro del campo receptor, los elementos centrales de los operadores A han sido definidos con signo opuesto al de los correspondientes elementos de los operadores B. En la figura 5.4 se observa que las Celdas bipolares comparten las mismas entradas y que, por tanto, la única diferencia es el signo opuesto de los elementos centrales de sus plantillas, como se muestra en las ecuaciones 5.4.

$$A_{ON} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, B_{ON} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, I_{ON} = 0$$

$$A_{OFF} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, B_{OFF} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 0 \end{pmatrix}, I_{OFF} = 0 \tag{5.4}$$

Resultados de la ejecución del modelo

Las Celdas CNN del modelo artificial han sido configuradas para procesar imágenes de tamaño 640×480 píxeles con 8 bits de precisión. Dado que el modelo biológico está formado por tres neuronas (1 célula horizontal y 2 células bipolares), el circuito artificial propuesto emulará, de forma equivalente, un total de 921.600 neuronas ($640 \times 480 \times 3$).

El modelo se ha probado con la imagen de test mostrada en la figura 5.5-a, su ejecución ha devuelto dos salidas: la respuesta del circuito ON (figuras 5.5-b) y la respuesta del circuito OFF (figura 5.5-c). La imagen de salida del circuito ON ha sido ampliada con el objeto de apreciar los detalles de las bandas de Mach que se producen entre las regiones con distinto brillo. Dichos detalles son mostrados en la parte superior de la figura 5.5-d. La ilusión óptica que representa las bandas de Mach, es consecuencia de la inhibición lateral que provoca la Celda horizontal sobre las Celdas bipolares y, por tanto, más apreciables conforme mayor

es la influencia de la región periférica sobre el campo receptor. En el sistema visual primario, la inhibición lateral permite mejorar la percepción del contraste, y representa una de las principales tareas de procesamiento espacial que realiza la retina.

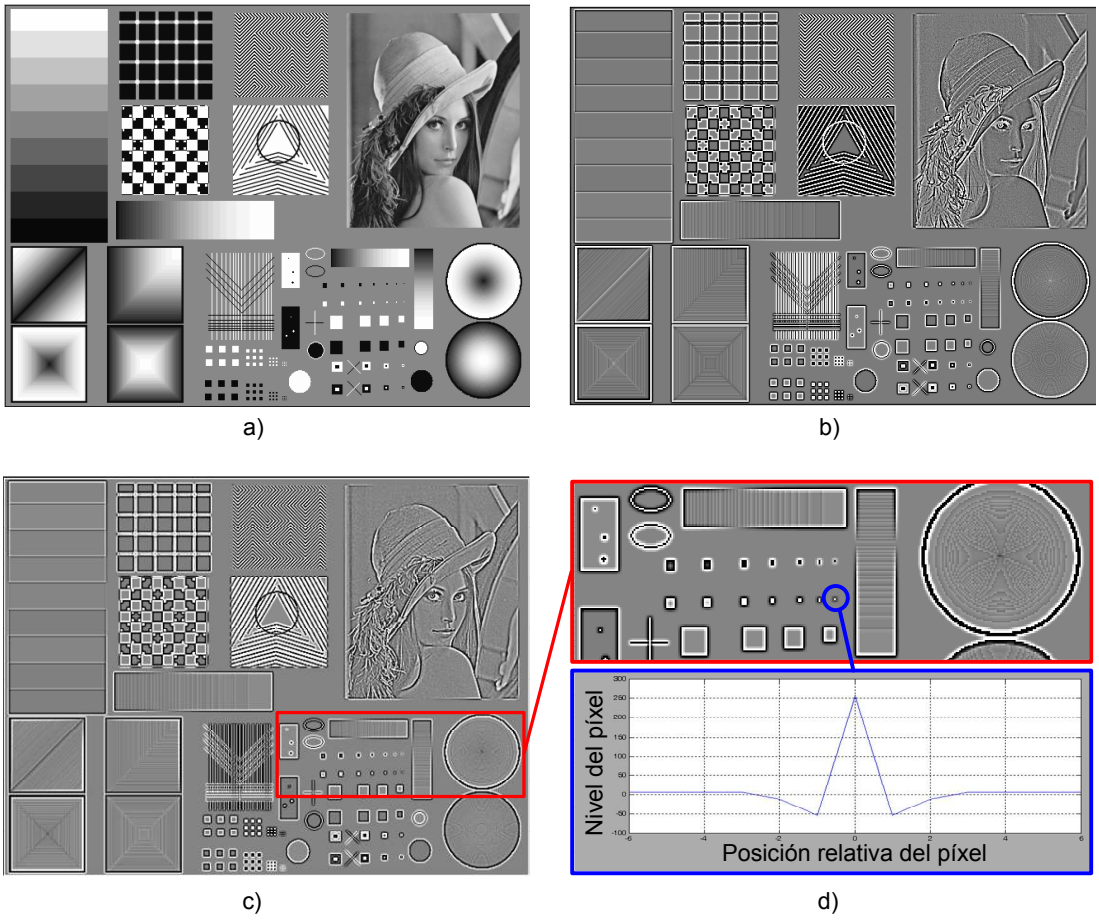


Figura 5.5: Resultados de ejecución del modelo. a) Imagen de entrada. b) Imagen de salida de la celda bipolar OFF. c) Imagen de salida de la celda bipolar ON. d) Detalles del gradiente de brillo detectado y respuesta del campo rector ON.

En la parte inferior de la figura 5.5-d, se muestra la respuesta de la Celda bipolar ON ante la influencia de un píxel a máximo brillo. En la curva obtenida, se observa que la mayor intensidad de la respuesta se produce cuando el estímulo aplicado coincide directamente con el centro del campo receptor (desviación cero con respecto a la posición del píxel). Además, cuando la posición del píxel es desplazada con respecto al centro, la curva va adquiriendo la forma de sombrero

mejicano del campo receptor ON. En la gráfica se aprecia que el valor mínimo de la curva aparece justo alrededor del píxel central y que, según la configuración del circuito considerado, el tamaño de la región de influencia coincide con la superficie establecida de 5×5 píxeles. En la figura 5.5-b se puede observar que la respuesta del campo receptor OFF es justamente la opuesta.

5.1.4. Aproximación a la sinapsis-I de la retina basada en redes CNN no lineales

El modelo que se propone a continuación tiene en cuenta la variabilidad de los circuitos neuronales de la retina en función de su posición con respecto al campo visual. El modelo incluye distintos circuitos, con diferentes grados de convergencia y conectividad, que representan diferentes regiones de la primera sinapsis de la retina. No obstante, dada la dificultad que supe emular en hardware una estructura continua, como la de la retina, se ha recurrido a una simplificación que considera a esta como una estructura heterogénea, formada por seis regiones diferentes. Estas regiones abarcan desde la región central de la retina (que denominaremos región R1) hasta la región periférica (región R6). La arquitectura hardware CNN que se propone para el modelo estará formada por seis circuitos diferentes, cada uno de ellos diseñado para emular una región distinta a partir de un campo receptor de tipo ON (Kolb 2003; Dacey et al., 2000; Packer y Dacey 2002)]. El desarrollo del campo receptor OFF es igual que el del campo receptor ON y, por simplicidad, no será considerado en esta aproximación.

La figura 5.6 muestra los seis circuitos CNN utilizado para aproximar la primera sinapsis de la retina. Los circuitos con mayor cantidad de Etapas representan a los circuitos neuronales con mayor conectividad y factor de convergencia y se corresponden con las regiones más periféricas de la retina. Los circuitos con menor número de Etapas tienen menor convergencia y se utilizan para modelar los circuitos de la retina central. La aproximación dispone de una entrada de datos por la que recibe secuencialmente el flujo de datos a procesar. La arquitectura ha sido diseñada para que todos los circuitos procesen la entrada en paralelo y obtengan el resultado al mismo tiempo. Para conseguir esto, las entradas U_{ij} de los circuitos más cortos han sido conectadas a las salidas U_{ij} de los circuitos más largos, lo que permite igualar el flujo de datos de los circuitos. El multiplexor

de la salida se encargará de seleccionar la respuesta que será visualizada en cada momento. La selección dependerá de la posición del píxel que se esté procesando y estará gestionada por una pequeña unidad de control.

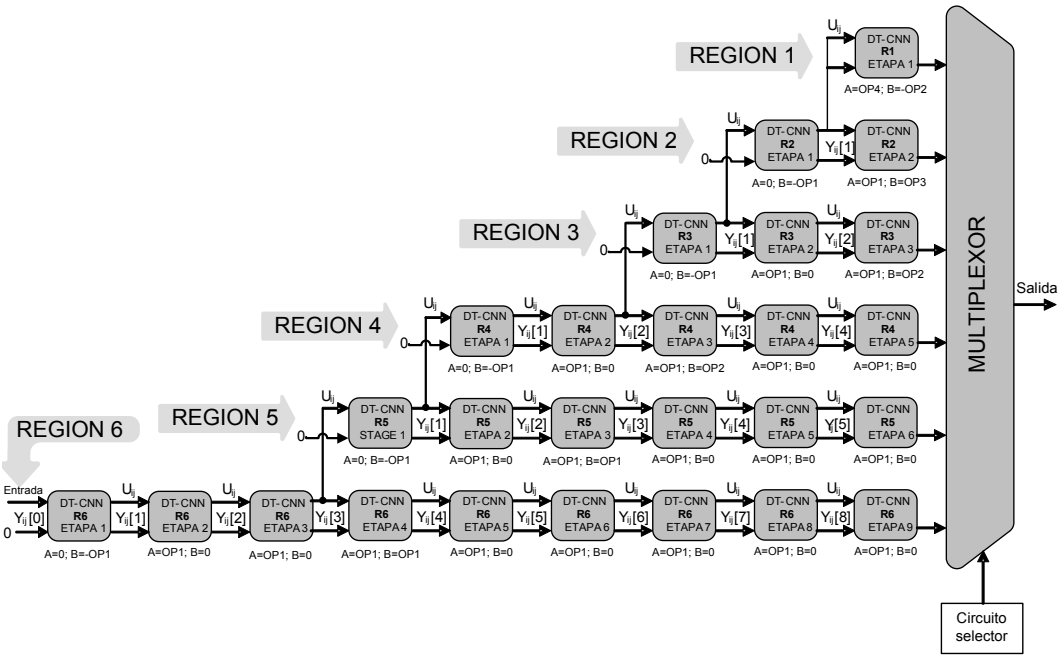


Figura 5.6: Arquitectura CNN con la seis regiones consideradas en la sinapsis-I.

Las plantillas de la arquitectura CNN propuesta representan las sinapsis de las células horizontales y bipolares de la retina, los valores de sus operadores darán forma a las curvas (picos y valles) de los campos receptores. Las expresiones 5.5 representan los cuatro tipos de operadores que han sido utilizados para desarrollar el modelo propuesto: el operador gaussiano OP1, los operadores de media OP2 y OP3, y el operador de identidad OP4.

$$OP1 = \begin{pmatrix} 0.042 & 0.117 & 0.3248 \\ 0.117 & 3 & 0.117 \\ 0.042 & 0.117 & 0.042 \end{pmatrix}, \; OP2 = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

$$OP3 = \begin{pmatrix} 0 & 1/2 & 0 \\ 1/12 & 1/1.5 & 1/12 \\ 0 & 1/12 & 0 \end{pmatrix}, \; OP4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{5.5}$$

Los operadores de media OP2 y OP3 han sido normalizados con el fin de evitar la saturación de la salida de los circuitos. Los pesos del operador OP1 siguen una distribución Gausiana y han sido calculados a partir de la ecuación 5.2. La combinación de estos cuatro operadores permite definir el tamaño de los campos receptores de las regiones, los cuales tendrán forma de sombrero mejicano al considerar que son el resultado de la superposición de dos superficies gaussianas, una excitadora y otra inhibidora.

Cada uno de los seis circuitos CNN propuestos presentan factores de convergencia diferente, es decir, el número de entradas que influyen directamente o indirectamente sobre la respuesta del circuito es diferentes en cada caso. Se han definido dos factores de convergencia: el factor que determina el número de fotorreceptores que influyen directamente sobre la Celda bipolar, denominado Factor1, y el factor que representa el número de fotorreceptores que influyen indirectamente (a través de la Celda horizontal) sobre la Celda bipolar, denominado Factor2. Los valores de estos factores son mostrados en el cuadro 5.1 para cada uno de los circuitos.

Cuadro 5.1: Factor de convergencia y número de Etapas de cada región

Región	Factor1	Factor2	Nº Etapas
R1	1	9	1
R2	5	49	2
R3	9	81	3
R4	25	121	5
R5	36	169	6
R6	49	225	9

En dicho cuadro se puede observar, por ejemplo, que la Celda bipolar de la región R6 está influenciada directamente por 49 entradas (7×7 píxeles), mientras que de forma indirecta, a través de la Celda horizontal, está influenciada por 225 entradas (15×15 píxeles). La implementación de los factores de convergencia se ha realizado incorporando a los circuitos el número de Etapas adecuado y determinando apropiadamente los valores de sus plantillas.

Resultados de la implementación y ejecución del modelo

La red CNN ha sido configurada para trabajar con imágenes de tamaño 640×480 píxeles y 8 bits de precisión. La estructura CNN está compuesta por 26 Etapas y permite emular un total de 3.686.400 neuronas biológicas, ya que se considera que cada uno de los seis circuitos esta formado una célula bipolar y una célula horizontal.

El esquema de la figura 5.6 ha sido implementado sobre una FPGA XC4VSX25-12 de Xilinx, utilizando la arquitectura Carthagonova con un multiplicador por unidad MAC. El cuadro 5.2 resume los resultados de velocidad y utilización de los recursos hardware de la FPGA. Como se observa, el mayor porcentaje de recursos consumidos corresponde al de Slices con un 70 %. El número total de operaciones aritméticas ejecutadas por la estructura asciende a 36 GOP/S (considerando 35 operaciones por Etapa), y la frecuencia máxima del circuito permite que la red CNN pueden trabajar en tiempo real a un máximo de 129 fps.

Cuadro 5.2: Resumen de los recursos empleado por la red CNN e información de tiempos, porcentajes referidos a la FPGA XC4VSX25-12.

Recursos	Unidades	% uso
Area (slices)	7204	70
Flip flops	9703	47
DSP48	52	40
BlockRAM	52	40
F. Max. CST (MHz)	436	-
F. Max. CLK (MHz)	39	-

La figura 5.7-a muestra la imagen de test utilizada para ajustar experimentalmente las plantillas de la red CNN. La figura 5.7-b muestra las imágenes obtenidas por cada uno de los seis circuitos del modelo (R1 en el lado izquierdo de la imagen y R6 en lado derecho). Como era de esperar, cada región realiza un procesamiento diferente que puede ser analizado a través de las bandas de Mach. La imagen proporcionada por el circuito R1 muestra las bandas de Mach más finas y la máxima resolución de detalle. Esto es coherente con el procesamiento realizado por

la fóvea, puesto que su menor convergencia e inhibición lateral provoca una mayor agudeza visual. La imagen obtenida por el circuito R6, encargado de emular la periferia de la retina, presenta mayor grado de convergencia y menor agudeza visual. En ella se observa que las bandas de Mach son más gruesas y que solamente se detectan las variaciones de contraste de mayor intensidad. Las imágenes de salida de los circuitos R2 a R5 muestran de forma progresiva, conforme se incrementan los factores de convergencia, un mayor engrosamiento de las bandas de Mach y una menor sensibilidad a los detalles finos.

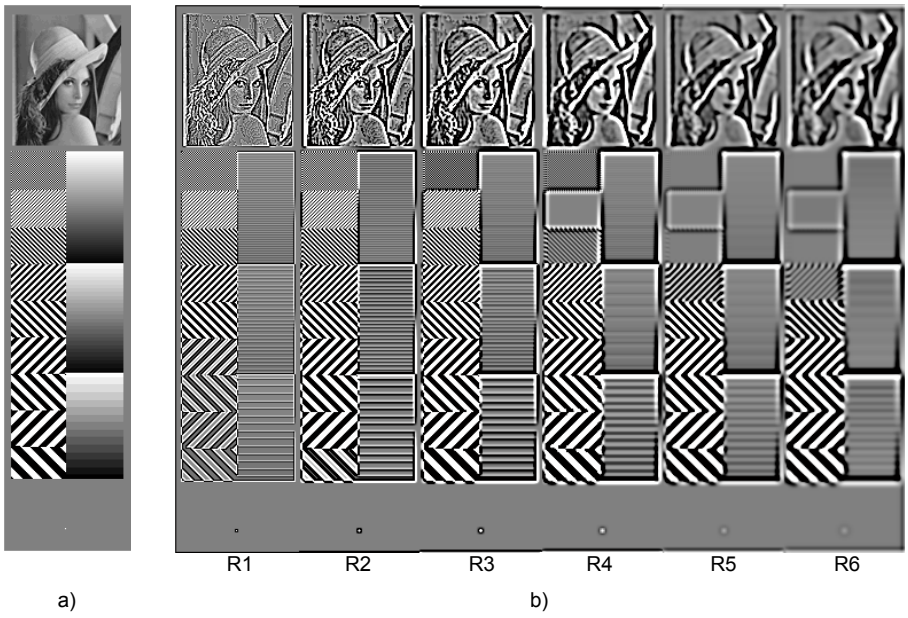


Figura 5.7: Imagen de test para el ajuste de las plantillas de la arquitectura. a) Imagen de entrada. b) Resultados de cada uno de los circuitos

La figura 5.8 muestra la respuesta de los seis circuitos ante una imagen formada por un píxel con máximo brillo. A medida que el píxel se desplaza de izquierda a derecha, la curva que se obtiene va tomando forma de sombrero mejicano, similar a la de los campos receptores de la retina. El máximo valor de la respuesta se consigue en el centro de la curva y la anchura de la distribución coincide con los factores de convergencia mostrados en el cuadro 5.1.

La imagen de la figura 5.9-a, de tamaño 640×480 píxeles, ha sido utilizado para comprobar la respuesta de cada uno de los circuitos. La figura 5.9-b muestra el efecto visual que se obtiene al combinar los diferentes resultados sobre una única

imagen, considerando que cada respuesta ocupa un anillo concéntrico diferente. El centro de la imagen se corresponde con la respuesta del circuito R1 y la parte más externa con la del circuito R6. Se puede observar que la parte central tiene mayor resolución visual, como ocurre en la fóvea, y que, conforme nos alejamos del centro, la resolución de detalle disminuye progresivamente. No obstante, a diferencia de lo que ocurre en la retina, la transición de una región a otra no es continua sino abrupta, debido a la hetereogeneida considerada en el modelo.

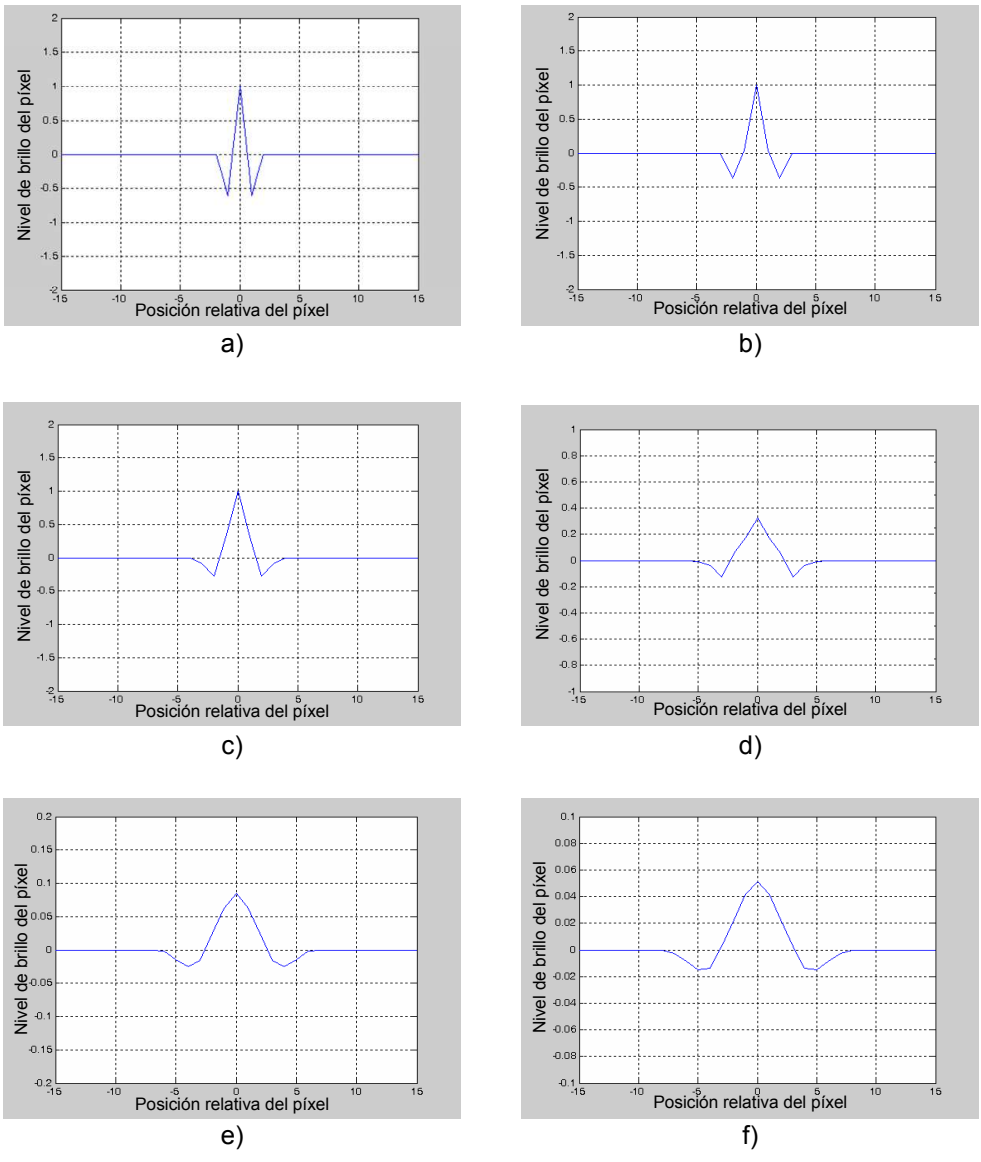
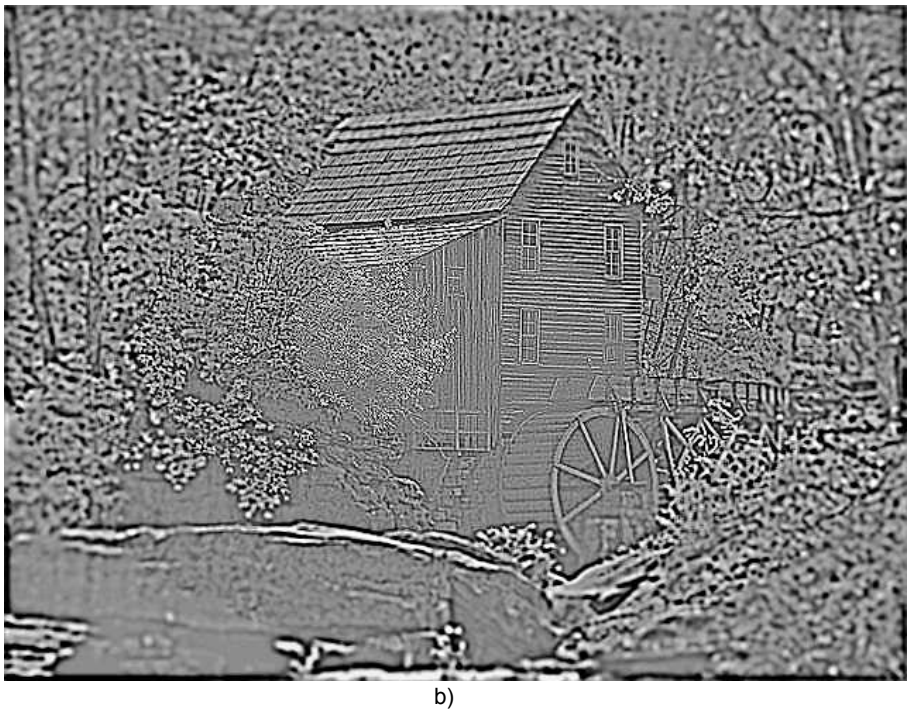


Figura 5.8: Respuesta de cada circuito ante una imagen de entrada con un píxel a máximo brillo (campo receptor equivalente).

a)



b)

Figura 5.9: Efecto visual obtenido combinando la salida de los seis circuitos del modelo artificial de la sinapsis-I de la retina. a) Imagen de entrada. b) Imagen formada con la respuesta de los seis circuitos.

5.1.5. Conclusiones

La arquitectura CNN desarrollada ha sido utilizada para implementar un modelo artificial de la primera sinapsis de la retina basado en campos receptores. La implementación del modelo ha permitido comprobar la influencia de la convergencia de la información y de la inhibición lateral de los circuitos artificiales en la detección del contraste. El desarrollo de esta aproximación ofrece un enfoque funcional del procesamiento espacial realizado por la retina y muestra el procedimiento de extracción de su comportamiento fisiológico básico para adaptarlo a los dispositivos hardware.

El modelo artificial de la primera sinapsis de la retina ha sido dividido en seis regiones diferentes, que han sido implementadas mediante seis circuitos electrónicos equivalentes basados en redes CNN.

A partir de los resultados de ejecución del modelo, se ha podido comprobar que la detección del contraste y la resolución de detalle obtenidas dependen de los factores de convergencia de las neuronas y de la inhibición lateral de los circuitos artificial, al igual que sucede en la retina. Para la visión foveal, emulada a partir de los circuitos con menor convergencia e inhibición lateral, se ha comprobado que tanto los detalles detectados en la imagen como la división del contraste han sido los más precisos. Conforme los objetos se localizan en posiciones más alejadas del centro de la imagen, la resolución de detalle disminuye y los cambios de contraste se detectan con menor precisión. Se ha comprobado, por tanto, que la inhibición lateral y el factor de convergencia pueden ser utilizados para controlar la detección de detalles y los cambios de contraste. Y que, su modificación, puede afectar a la detección de las bandas de Mach, y por tanto, a la percepción del sistema visual.

Es evidente que la complejidad del sistema visual no puede ser explicada a partir de circuitos tan sencillos como los propuestos por estos modelos. Hay que tener en cuenta, que el procesamiento realizado en la primera sinapsis de la retina es mucho más complejo y que solamente representa una pequeña parte del complejo conjunto de tareas que realiza el sistema visual primario. Por consiguiente, los resultados obtenidos deberán ser tratados como aproximaciones, que tendrán que ser completados con estructuras neuronales de mayor complejidad para poder desarrollar modelos más completos y realistas del sistema visual.

5.2. Sistema embebido multi-FPGA para el desarrollo de aplicaciones de vídeo en tiempo real basadas en redes CNN

La mayoría de las aplicaciones basadas en redes CNN suelen estar relacionadas con el procesamiento de imágenes. El diseño de sistemas de desarrollo que permitan la ejecución de estas aplicaciones en tiempo real, pasa por resolver los problemas asociados con sus exigentes requisitos de velocidad.

En los últimos años han visto la luz diversos sistemas embebidos destinados a desarrollar aplicaciones de procesamiento en tiempo real basadas redes CNN. Entre los más recientes destacan: un sistema (Voroshazi et al., 2008) que hace uso de la arquitectura Falcon (Nagy y Szolgay 2003), el sistema ACE16KDS (Carranza et al., 2005) que utiliza la arquitectura ACE16K (Rodriguez-Vazquez et al., 2005) y un sistema (Mika et al., 2008) basado en la arquitectura Helsinki (Paasio et al., 2002). En las siguientes secciones se describe un sistema embebido, basado en la arquitectura Carthagonova, cuyo objetivo es facilitar el desarrollo de aplicaciones de procesamiento de vídeo en tiempo real que requieran el uso de redes CNN complejas y de gran tamaño.

El sistema está basado en una arquitectura de cómputo distribuida, fácilmente ampliable, que permite la emulación de las redes CNN no lineales multi-capa de forma rápida y sencilla. El sistema ha sido diseñado para ejecutar de manera autónoma y versátil aplicaciones de procesamiento de vídeo en tiempo real, e incorpora algunos aspectos novedosos. En primer lugar, la arquitectura distribuida del sistema está compuesta por múltiples módulos de procesamiento basados en FPGAs, que permiten la expansión del sistema y su adaptación a múltiples aplicaciones. En segundo lugar el sistema ha sido diseñado para utilizar la arquitectura Carthagonova y aprovechar sus características para desarrollar aplicaciones con redes CNN complejas, obteniendo el máximo rendimiento de los recursos de las FPGAs. La modularidad, flexibilidad y prestaciones de la arquitectura Carthagonova favorecen la ampliación del sistema y dan soporte a un procesamiento eficiente. El sistema hereda la jerarquía y el esquema de procesamiento de la arquitectura Carthagonova, lo que permite desarrollar estructuras CNN complejas a partir de una sencilla organización de los componentes en niveles de abstracción.

Como se muestra en el esquema de la figura 5.10, el sistema esta formado por varias tarjetas de procesamiento interconectadas entre sí a través de una interfaz de expansión (IE) compartida. Cada tarjeta, denominada Módulo de Procesamiento (MP), incluye: una FPGA, con la red CNN o parte de ella, y el conector de la interfaz de expansión. La capacidad de cómputo total del sistema viene determinada por el número de MP utilizados y por la cantidad de recursos disponibles en las FPGAs. Las principales características del sistema se resumen en los siguientes puntos:

- El sistema consiste en una arquitectura de cómputo distribuida, modular y ampliable, formada por varios módulos de procesamiento independientes, que dan versatilidad al sistema y que permiten su adaptación a aplicaciones de diferente tamaño.

- El sistema ha sido diseñado para desarrollar aplicaciones de procesamiento de video en tiempo real que requieran la utilización de redes CNN complejas y de gran tamaño.

- La arquitectura CNN Carthagonova, utilizada por el sistema, da soporte a la expansión del sistema y permite organizar jerárquicamente la estructura interna de las redes CNN.

- Los componentes hardware de la arquitectura CNN han sido diseñados para conseguir un compromiso equilibrado entre área y velocidad, y aprovechar eficientemente los recursos internos de las FPGAs.

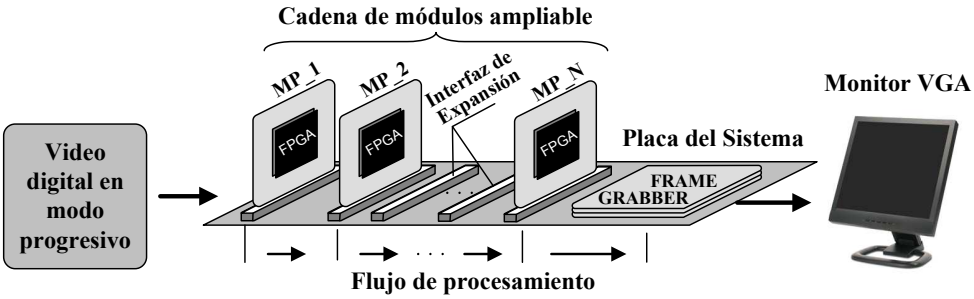


Figura 5.10: Esquema del sistema distribuido propuesto.

5.2.1. Descripción y organización del sistema

El sistema ha sido desarrollado para emular redes CNN de gran tamaño a partir de una arquitectura de cómputo distribuida compuesta por múltiples Módulos de Procesamiento (MP) basados en FPGAs. Esta arquitectura modular facilita la ampliación del sistema y el incremento de la capacidad de cómputo. Las redes CNN pueden ser distribuidas sobre los diferentes MP con el fin de incrementar la versatilidad y el rendimiento computacional del sistema.

La plataforma hardware del sistema incluye además una Interfaz de Expansión (IE), que ha sido diseñada para conectar los MPs en cascada y facilitar la incorporación de nuevos módulos en el sistema. Los MPs trabajan en flujo de datos y soportan las características del procesamiento, secuencial y en tiempo real, de la arquitectura Carthagonova. La IE tiene como objetivo recibir y transmitir la información de los MPs y ofrecer la suficiencia versatilidad para permitir la ampliación del sistema de forma cómoda y sencilla. Todos los MPs procesan la información de vídeo en flujo de datos y proporcionan una salida compatible con la entrada del siguiente módulo de cadena. Cada MP es implementado en una tarjeta con FPGAs, donde se alojan la red CNN o parte de ella, e incorpora el conector de la IE. El sistema incluye un *frame grabber* que corrige el desfase entre la información de vídeo procesada y los sincronismos introducidos en la entrada.

Como muestra la figura 5.11, una fuente de vídeo digital, en modo progresivo, debe ser conectada al sistema ocupando la primera posición de la cadena, usando el primer conector de la IE (IE 1). Dicha fuente proporciona la información de vídeo que procesarán los MPs y las señales de control que serán propagadas a lo largo de toda la cadena. Estos sincronismos son utilizados para poner en marcha los MPs y sincronizar el funcionamiento del *frame grabber*. Conforme el flujo de vídeo es procesado, y propagado a lo largo de la cadena, los módulos son activados entrando en juego en el procesamiento. El retraso de la información de vídeo a la entrada del *frame grabber* dependerá del número de Etapas CNN conectadas en cascada a lo largo de la cadena de MPs. La latencia de los módulos, introducirá un desfase entre la información de vídeo y las señales de sincronismos, que será corregido por el *frame grabber*. El *frame grabber* proporciona una salida compatible con el estándar de video VGA y debe ser conectado en el último conector de la IE (IE N+2). El *frame grabber* podrá ser eliminado del sistema

cuando el retraso en los MPs coincida con el periodo de refresco de la imagen, como se indicó en la sección 4.11.

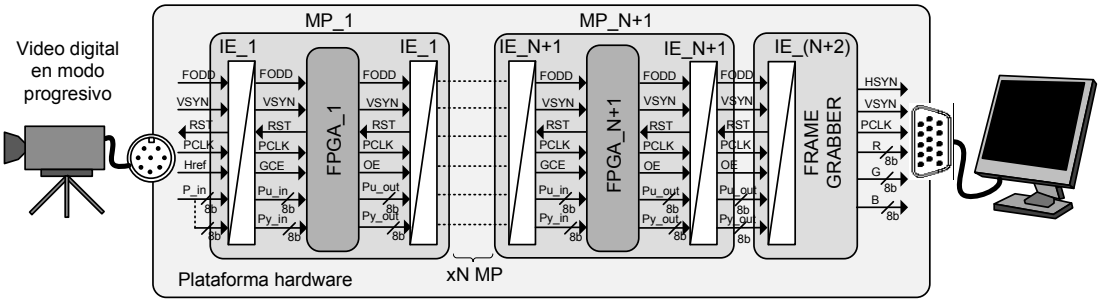


Figura 5.11: Plataforma hardware del sistema.

En las secciones siguientes se desarrolla una particularización del sistema cuyo objetivo es realizar un prototipo para su verificación.

5.2.2. Módulos de procesamiento e interfaz de expansión

Los componentes hardware que definen a los MPs y la IE han sido adaptados a las características particulares de la arquitectura CNN Carthagonova. A través de la IE, la fuente de vídeo proporcionará a los módulos (MP o *frame grabber*), la información de vídeo y las señales de control necesarias para iniciar el funcionamiento. Mientras que los módulos conectados a la interfaz procesarán la información recibida y la propagarán, junto con las señales de control, a los siguientes módulos de la cadena.

La figura 5.12 muestra los puertos de entrada/salida de la IE del sistema. El MP mostrado en el ejemplo representa a un módulo formado por n-Redes CNN conectadas en cascada. En general, la complejidad y el tamaño de la estructura CNN incluida en los MPs dependerán de los requerimientos de cada aplicación y de los recursos disponibles en la FPGA.

Para dar soporte al procesamiento flujo de datos, la IE ha sido diseñada a partir de un esquema simple basado en pares de puertos complementarios: un puerto de entrada y otro de salida, ambos asociados al mismo flujo de datos. Esta característica permite que los MPs puedan ser conectados y desconectados del sistema, de forma sencilla, teniendo en cuenta sólo dos consideraciones: la

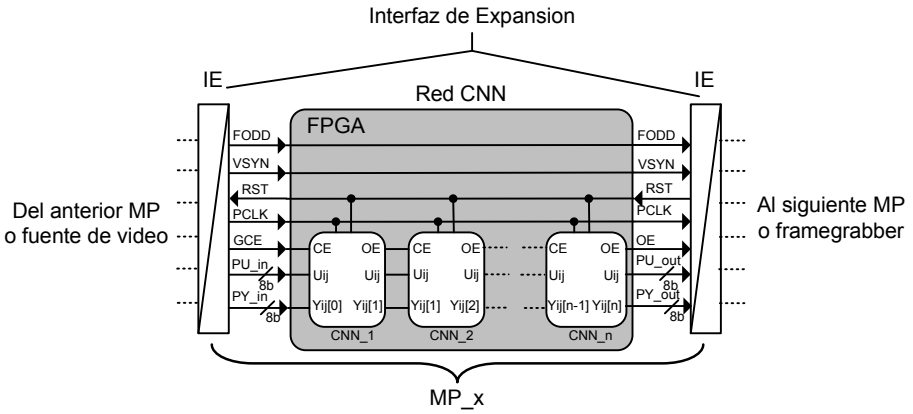


Figura 5.12: Modulo de procesamiento e interfaz de expansión del sistema.

primera, que los MPs tendrán que ser conectados y desconectados del sistema cuando esté apagado, y la segunda, que los pares de puertos de los conectores no utilizados tendrán que puentearse para permitir el flujo de información hacia los siguientes módulos. Este esquema permite sobredimensionar el número de conectores del sistema y utilizar solamente aquellos que sean necesarios. Una vez que los MPs han sido conectados al sistema, las FPGAs que incorporan podrán ser configuradas con la estructura CNN adecuada.

La información de vídeo que es propagada a lo largo de la cadena es recibida por los IE a través de dos puertos de entrada: el puerto $\mathrm{PU_{in}}$, que recibe la entrada principal del sistema sin procesar, y el puerto $\mathrm{PY_{in}}$, que recibe la información de vídeo procesada por el módulo anterior. En el interior de cada MP, la información de vídeo de ambas entradas es procesada por la red CNN. El resultado del procesamiento es devuelto a la IE a través del puerto de salida $\mathrm{PY_{out}}$, que está conectado a la entrada $\mathrm{PY_{in}}$ del siguiente módulo de la cadena. El otro puerto de salida $\mathrm{PU_{out}}$, propaga el flujo de datos de la entrada principal del sistema al puerto $\mathrm{PU_{in}}$ del siguiente módulo. Dicho flujo de datos llegará a todos los puertos PU del sistema sin ser procesado, aunque sí con un retraso que dependerá del número de Etapas por las que se haya propagado.

Otros puertos de la IE son: PCLK (reloj de píxel), RST (reset), GCE (habilitación global del circuito), OE (habilitación de la salida), VSYN (señal de inicio) y FODD (modo progresivo). El puerto de entrada GCE se utiliza para habilitar el funcionamiento de los módulos conectados a la interfaz. El puerto de salida OE será activado por la red CNN cuando las salidas de los MPs ($\mathrm{PU_{out}}$ y $\mathrm{PY_{out}}$)

comiencen a producir datos válidos. Para mantener la coherencia en el flujo de las señales de control, el puerto de salida OE del conector anterior debe conectarse al puerto de entrada GCE del conector siguiente. Las señales recibidas por los puertos VSYN y FODD son generados por la fuente de vídeo principal y propagadas a lo largo de la cadena de MPs hasta el *frame grabber*. Estas señales son utilizadas, respectivamente, para iniciar la adquisición de imágenes en el *frame grabber* y para comprobar si la fuente de vídeo proporciona la información en modo progresivo o en modo entrelazado.

5.2.3. Prototipo del sistema

En la figura 5.13 se muestra el esquema del prototipo del sistema y la arquitectura interna del *frame grabber*. Como se observa, la fuente de vídeo ha sido conectada al sistema utilizando el primer conector de la interfaz de expansión. La fuente de vídeo proporciona, en modo progresivo, las imágenes de vídeo que son procesadas por los MPs y las señales de sincronismos que necesita el sistema, concretamente, el sincronismo vertical (Vsyn) y el sincronismo horizontal (Href). El sincronismo Vsyn se conecta al puerto VSYN y se utiliza para habilitar el *frame grabber* y sincronizar el periodo de refresco de las imágenes. El sincronismo Href se conecta al puerto de entrada GCE del primer módulo (MP1) y sirve para habilitar el procesamiento de los MPs.

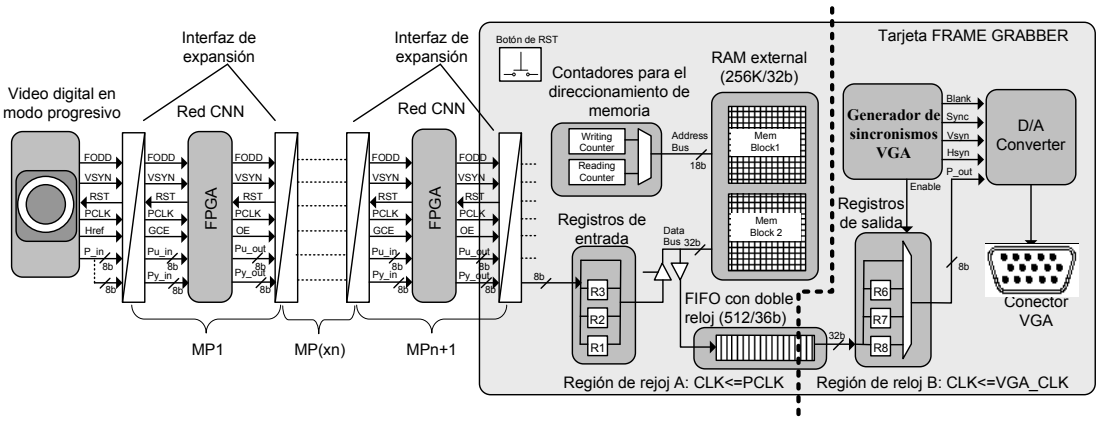


Figura 5.13: Aquitectura completa del prototipo y esquema interno del *frame grabber*.

El *frame grabber* se conecta al sistema utilizando el último conector de la interfaz de expansión, su entrada de vídeo, por tanto, es proporcionada por el último módulo de procesamiento (MPn+1). La salida de vídeo generada por el *frame grabber* es compatible con el estándar de vídeo VGA. La arquitectura interna del *frame grabber* ha sido dividida en dos regiones del reloj con el fin de poder trabajar simultáneamente con la información de vídeo de la entrada, sincronizada con el reloj PCLK, y con la información de vídeo de la salida, sincronizada con el reloj VGACLK. Para conseguir que el *frame grabber* funcione correctamente, la frecuencia del sincronismo vertical de la entrada (VSYN) debe ser igual, o múltiplo, a la frecuencia del sincronismo vertical del estándar VGA industrial (60 Hz).

Cuando la información de vídeo es recibida por el *frame grabber*, los datos son almacenados en una memoria RAM externa con capacidad para 2 imágenes. Dicha memoria ha sido dividida en dos bloques para separar los procesos de lectura y escritura; en uno bloque se escribe la imagen de entrada, mientras que en el otro se lee la imagen de salida. Cuando la imagen de entrada termina de ser almacenada, el hardware conmuta los modos de acceso y el bloque de escritura pasa a lectura y el de lectura a escritura.

Los procesos de escritura y lectura de los bloques de memoria se realizan llevando a cabo 1 ciclo de escritura por cada 3 ciclos de lectura. En cada ciclo de escritura se almacenan en memoria 4 píxeles consecutivos de la imagen de entrada. Esto se consigue concatenando y almacenando previamente, en un banco de registros, los últimos 4 píxeles recibidos en la entrada del *frame grabber*. Durante los ciclos de lectura, la imagen de salida es extraída de la memoria en palabras de 32 bits, que a continuación son almacenadas en una memoria FIFO de doble puerto con capacidad para 512 palabras.

La memoria FIFO es utilizada como buffer para desacoplar las dos regiones del reloj del *frame grabber*. A través de su puerto de entrada, sincronizado con el reloj PCLK, se introducen las palabras leídas de la memoria RAM, mientras que a través de su puerto de salida, sincronizado con el reloj VGACLK, se extraen las palabras para llevarlas a un demultiplexor y separarlas en 4 píxeles independientes. Dado que sobre la RAM se realizan tres ciclos de lectura por cada ciclo de escritura, la FIFO dispone en todo momento de información suficiente para cargarla al demultiplexor.

El *frame grabber* incorpora un circuito generador de sincronismos, que controla la temporización del demultiplexado y genera las señales de sincronismo del estándar VGA. Finalmente, el flujo de vídeo pasa por un convertidor digital/analógico (DAC), cuyas salidas, junto con las señales de sincronismos, son llevadas al conector VGA.

5.2.4. Realización del prototipo

Un prototipo del sistema, formado por dos MPs, ha sido realizado utilizando tres placas basadas en FPGA. El objetivo de esta realización es verificar la metodología de diseño del sistema y comprobar su funcionalidad. El prototipo está formado por una cámara de vídeo digital (OV7620), dos MP, un *frame grabber* y un monitor VGA. La cámara ha sido configurada para trabajar en modo progresivo y proporcionar imágenes digitales en escala de grises de 640×480 píxeles y 8 bits de precisión a 30 fps. Los MP cuentan con unas dimensiones muy reducidas (7x8 cm) y han sido implementados con la tarjeta PICO E-12 (Picocomputing, 2011), que incluye una FPGA XC4VFX12 de Xilinx con 32 DSP48 y 36 BlockRAM. El *frame grabber* ha sido implementado con la tarjeta de bajo coste SP3-EVL1500 (Avnet, 2011) que incorpora una FPGA XC3S1500 de Xilinx, 1 MB de memoria RAM externa y un convertidor digital analógico (DAC) ADV7125 de *Analogic Device*.

La funcionalidad del sistema ha sido evaluada utilizando una red CNN de 3 Capas con plantillas variantes en el espacio. Las 3 Capas aplicadas sobre la imagen están conectadas en cascada e incluyen una Celda de 1, 5, y 5 Etapas Carthagonova respectivamente. Aunque esta red CNN puede ser implementada en un único MP, las Capas han sido repartidas entre los dos MPs con el fin de verificar la capacidad de computación distribuida del sistema. Las dos primeras Capas han sido implementadas sobre el MP1, mientras que la tercera ha sido implementada sobre el MP2.

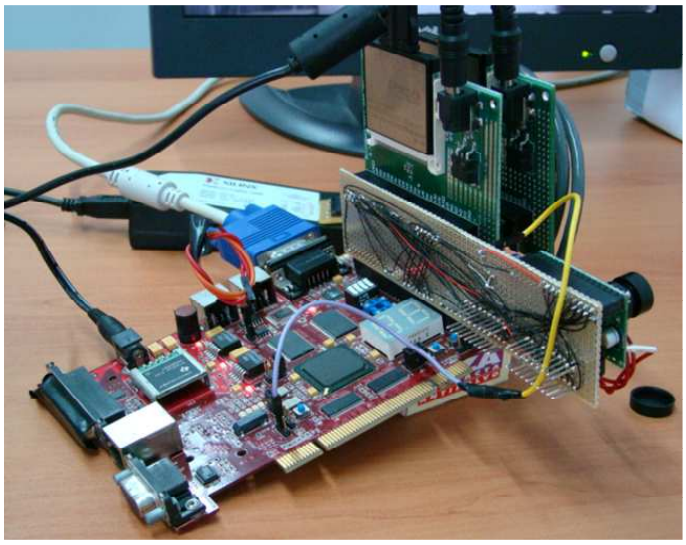
El algoritmo seleccionado realiza un procesamiento variante en el espacio, que depende del cuadrante de la imagen. Para ello, las Capas de la red CNN han sido configuradas con cuatro conjuntos de plantillas diferentes que son seleccionadas en función de la posición del píxel que se está procesando. Los píxeles del cuadrante superior izquierdo son procesados con las siguientes plantillas: difusión, detección

de bordes e identidad, en ese orden. Los píxeles del cuadrante superior derecho utilizan las plantillas: inversión, detección de bordes y difusión. El cuadrante inferior izquierdo es procesado por tres plantillas iguales: identidad. Y, finalmente, el cuadrante inferior derecho también es procesado por tres plantillas iguales: difusión. La figura 5.14 muestra la implementación del prototipo y el resultado del procesamiento en tiempo real, tras aplicar la red ML-CNN a la escena captada por la cámara.

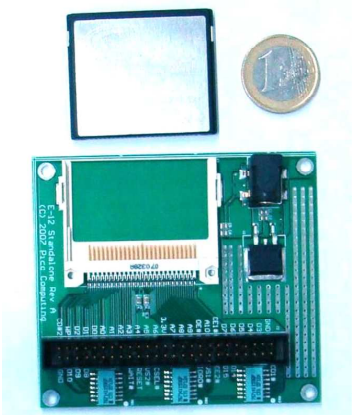
La fotografía de la figura 5.14-c fue tomada con un tiempo de exposición corto para conseguir capturar la caída de la pelota. Como se observa, el monitor VGA muestra un cierto retraso en la caída, con respecto a la escena real, como consecuencia del retraso introducido por *frame grabber*. En la esquina superior derecha de la figura se incluye una foto de la tarjeta FPGA usada por los MPs.

La capacidad de cómputo del prototipo está limitada por el número de MPs y las FPGAs utilizada (XC4VFX12). El número máximo de Etapas Carthagonova que pueden ser implementadas en este prototipo es de 32 Etapa, debido a que en estas FPGAs únicamente hay disponibles 32 DSP48. No obstante, hay que recordar que el prototipo puede ser fácilmente ampliado añadiendo nuevos conectores de expansión y nuevos MPs.

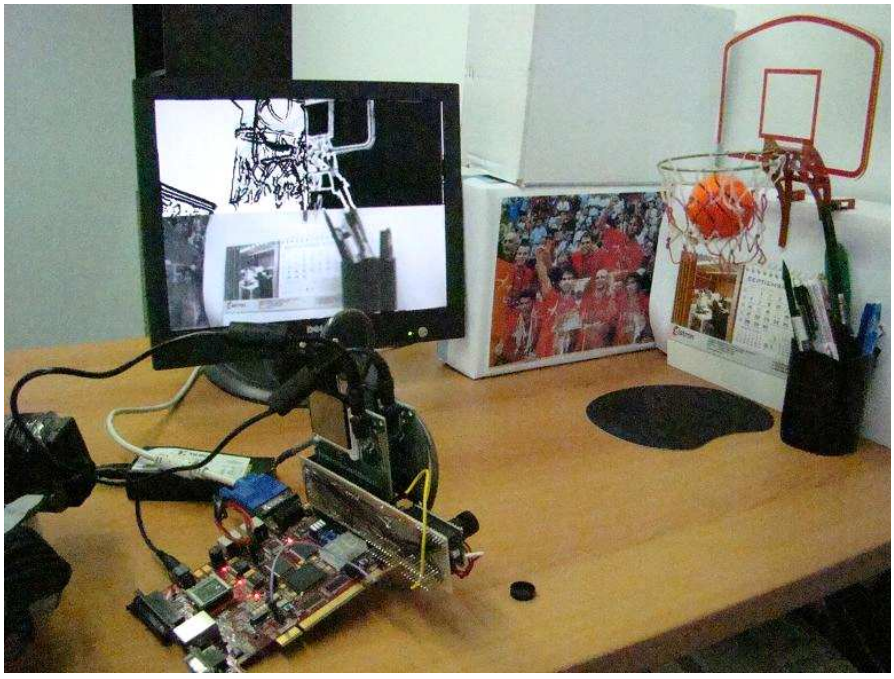
Cabe concluir que el sistema ha sido diseñado para desarrollar aplicaciones de procesamiento de vídeo en tiempo real y heredar la modularidad, la versatilidad y la ampliabilidad de la arquitectura CNN. De esta manera, el sistema podrá acomodar, de forma rápida y sencilla, aplicaciones basadas en redes CNN de distinto tamaño y complejidad. La arquitectura CNN y el sistema de cómputo distribuido propuestos representan una plataforma valiosa para el desarrollo de aplicaciones con redes CNN, mediante una metodología de diseño jerárquica basada en componentes eficientes previamente diseñados y verificados.



a)



b)



c)

Figura 5.14: Fotografías del sistema. a) Prototipo compuesto por la tarjeta del *frame grabber*, 2 módulos de procesamiento (MP), la cámara digital OV7620 y un monitor de vídeo VGA. b) Detalle del módulo de procesamiento y del conector de la interfaz de expansión. c) Fotografía del prototipo y banco de pruebas.

Conclusiones, líneas futuras y aportaciones

Conclusiones

En esta Tesis se ha propuesto el diseño y la implementación de una arquitectura hardware para la emulación en tiempo real de redes CNN y de un sistema de cómputo distribuido, que permiten el desarrollo de aplicaciones basadas en redes CNN complejas y de gran tamaño.

Durante el diseño de la arquitectura hardware se ha puesto especial interés en satisfacer aquellos requisitos del diseño que mejoran las prestaciones de velocidad de la arquitectura y su consumo de recursos hardware. Para organizar y simplificar el desarrollo de aplicaciones complejas, y hacer mas eficiente la implementación de las redes CNN, el hardware ha sido definido a partir de una arquitectura externa y una arquitectura interna. La arquitectura externa facilita la organización y el diseño de las redes CNN y sus componentes de mayor nivel, mediante el uso de una estructura jerárquica basada en un conjunto de componentes básicos previamente diseñados. La arquitectura interna, que define el componente de menor nivel de la arquitectura externa: la Etapa, ha sido definida e implementada de varias maneras usando diferentes técnicas y lenguajes de descripción hardware. El objetivo ha sido encontrar la mejor compromiso entre el consumo de recursos hardware y la velocidad de procesamiento de la arquitectura.

Por otra parte, el sistema de cómputo distribuido ha sido diseñado para utilizar la arquitectura hardware CNN y desarrollar aplicaciones embebidas basadas

en redes CNN de gran tamaño y complejidad. Las principales características que distinguen al sistema son: sencillez, modularidad, escalabilidad, versatilidad y flexibilidad, todas ellas imprescindibles para facilitar la ampliación del sistema y adaptarlo al mayor número posible de aplicaciones.

Las principales aportaciones del trabajo se resumen a continuación:

- Se ha llevado a cabo una revisión del paradigma de las redes CNN, abarcando los conceptos básicos, sus principales características, su arquitectura clásica y las variantes al modelo estándar más utilizadas. Se han revisado diferentes soluciones para la realización de las redes CNN, considerando sus principales características, ventajas y desventajas. Se ha prestado especial atención a las implementaciones hardware de propósito específico de las CNN-UM sobre circuitos ASIC analógicos, así como a la emulación de las redes CNN discretas sobre dispositivos hardware reconfigurables, estas últimas analizadas con mayor detalle.

- Se ha definido una arquitectura hardware con la que es posible diseñar redes CNN complejas siguiendo un esquema jerárquico, modular y sencillo. Para llevar a cabo el desarrollo de esta arquitectura se han propuesto diferentes métodos de discretización del modelo de la red CNN que han sido simulados y analizados. Como resultado de este estudio, se ha concluido que la aproximación discreta de Euler es el modelo más preciso a igual cantidad de recursos de cómputo. A partir de esta aproximación se ha planteado un algoritmo de procesamiento que define el comportamiento de la red CNN, el cual ha sido paralelizado y adaptado a las características de una implementación hardware. Este proceso de adaptación a dado lugar a una arquitectura hardware dedicada a emular en tiempo real el funcionamiento de la red CNN. Para simplificar el desarrollo, la arquitectura se ha dividido en arquitectura externa y arquitectura interna.

  La arquitectura externa, compuesta por diferentes componentes jerárquicos de alto nivel, ha sido diseñada para facilitar y simplificar el desarrollo de redes CNN complejas y de gran tamaño. El modo de procesamiento en flujo de datos, que caracteriza a esta parte de la arquitectura, reduce las comunicaciones entre los componentes y permite la ejecución paralela y distribuida de las redes CNN. La versatilidad y flexibilidad de los componentes de la ar-

quitectura externa posibilitan la ampliación de la capacidad de cómputo del sistema y favorecen su adaptación a los requerimientos de las aplicaciones.

El componente de menor nivel de la arquitectura externa, denominado Etapa, ha sido definido en hardware a partir de diferentes arquitecturas internas: la arquitectura Carthago y la arquitectura Carthagonova. Dichas arquitecturas son el resultado de la búsqueda del mejor compromiso entre la simplicidad, la eficiencia en el consumo de los recursos de cómputo y la mejora en la velocidad de procesamiento. En lo que respecta al consumo de área, ambas arquitecturas han sido diseñadas para hacer un uso eficiente de los recursos internos de las FPGAs. Para conseguir las máximas prestaciones de velocidad, la estrategia utilizada ha sido aplicar técnicas de super-segmentación al circuito, lo que ha permitido separar la Etapa en distintas regiones de reloj que han sido optimizadas de forma independiente.

- La implementación hardware de las arquitecturas Carthago y Carthagonova se ha realizado utilizando diferentes técnicas de sincronización y distintos lenguajes y niveles de descripción hardware. Las diferentes combinaciones empleadas han sido: sincronización *Selft-timed* con lenguaje VHDL a bajo nivel, sincronización convencional y lenguaje VHDL a nivel RTL estructural, y lenguaje a nivel de sistema (ESL, Impulse-C) sobre un computador reconfigurable de altas prestaciones (HPRC) tipo DS1002.

  Los resultados obtenidos por las implementaciones en VHDL a bajo nivel y mediante Impulse-C no han sido todo lo satisfactorias que se esperaba. En el primer caso, debido a las dificultades de la herramienta de síntesis para distribuir las macros de componentes sobre la FPGA, y en el segundo caso, debido a que las herramientas de síntesis a nivel de sistema todavía no han alcanzado la madurez suficiente como para maximizar el rendimiento de los recursos internos de las FPGAs. En la actualidad la intervención del diseñador sigue siendo necesaria para adaptar los algoritmos al hardware, así como un estilo de codificación cuidadoso para conseguir resultados aceptables.

  La implementación de la arquitectura Carthagonova en VHDL a nivel RTL estructural ha demostrado ser la solución más exitosa al proporcionar mejores prestaciones. En particular, la arquitectura super-segmentada basada en un multiplicador por unidad MAC ofrece el máximo aprovechamiento de

los recursos internos de las FPGAs y excelentes prestaciones en cuanto a velocidad de procesamiento.

- A partir de la arquitectura CNN propuesta se ha desarrollado un modelo artificial de la primera sinapsis de la retina, el cual considera los principales factores de convergencia y de inhibición lateral que influyen en los campos receptores de las neuronas biológicas. La respuesta obtenida por el modelo, tras ser analizado con diferentes imágenes de test, ha mostrado que la aproximación propuesta permite detectar los cambios de contraste en las imágenes y emular el efecto de las bandas de Mach que ocasiona el procesamiento espacial de la retina.

- Para validar el funcionamiento de la arquitectura CNN se ha diseñado un sistema de cómputo distribuido, basado en múltiples FPGAs, que permite emular en tiempo real el procesamiento realizado por una red CNN compleja y de gran tamaño. La realización del sistema ha sido laboriosa debido a que todos los componentes de la red CNN, las comunicaciones, el flujo de datos, las señales de control, etc. han tenido que ser adaptados a los requerimientos de una implementación particular, en la que se han tenido en cuenta todos los aspectos relacionados con el desarrollo de bajo nivel, incluyendo por ejemplo, la configuración de la fuente de vídeo, la adaptación de los niveles de tensión y la creación de componentes auxiliares complejos como el *frame grabber*.

- Se ha realizado un prototipo del sistema que permite emular redes ML-CNN de forma modular y versátil, y que puede utilizarse de manera autónoma para desarrollar aplicaciones de procesamiento de vídeo en tiempo real. El prototipo dispone de 2 módulos de procesamiento, y su funcionamiento ha sido validado experimentalmente llevando a cabo la emulación de una red CNN multi-capa de 3 Capas con plantillas variantes en el espacio.

  Los resultados obtenidos por el prototipo han sido satisfactorios desde varios puntos de vista. Por un lado se ha comprobado que tanto la arquitectura CNN como el sistema de cómputo distribuido funcionan correctamente y que, por tanto, es posible validar la metodología de diseño utilizada. Se ha demostrado además, que el sistema es suficientemente versátil y flexible como para ser utilizado como un excelente banco de pruebas en el desarrollo

de aplicaciones basadas en redes CNN complejas, que requieran el uso eficiente de los recursos hardware y trabajar en tiempo real. La modularidad del sistema permite la rápida ampliación del prototipo y sacar provecho de la organización jerárquica de los componentes hardware de la arquitectura CNN.

Líneas futuras

Las cualidades y características de la arquitectura CNN como, por ejemplo, su modularidad, flexibilidad, escalabilidad y eficiencia en área y velocidad, abren diferentes posibilidades en cuanto al desarrollo de futuras líneas de trabajo. A continuación se indican aquellas que se han considerado más interesantes.

- En primer lugar, sería conveniente llevar a cabo las dos modificaciones propuestas a la arquitectura interna de la Etapa Carthagonova, y analizar sus efectos sobre la velocidad de procesamiento. La primera modificación consistiría en eliminar el camino crítico que aparece entre la sub-etapa S2 y sub-etapa S3, para lo cual se debería incluir un registro adicional en la salida de la unidad MAC y modificar la unidad de control de la Etapa. La segunda modificación consistiría en eliminar el ciclo de reset de las unidades MAC y reducir el número de ciclos de reloj que son necesarios para completar el procesamiento. En esta última modificación, lo más interesante a analizar sería comprobar el efecto que produce la realimentación externa del DSP48 de la unidad MAC sobre la frecuencia de funcionamiento del circuito.

- Otra línea de trabajo interesante podría ser el desarrollo de una herramienta software gráfica, basada en los componentes y la jerarquía de la arquitectura externa propuesta, que posibilitará el diseño y la modificación gráfica de aplicaciones que requieran el uso de redes CNN. Esta herramienta, que podría realizarse mediante el entorno de desarrollo *System Generator*, permitiría describir estructuras complejas basadas en redes CNN, de forma rápida y sencilla, y a continuación obtener el código VHDL correspondiente en función de las especificaciones del diseño.

- Por otro lado, la primera experiencia con las plataformas HPRC y las herramientas ESL ha mostrado la viabilidad de estas para llevar a cabo el prototipado rápido de aplicaciones basadas en redes CNN, combinando la flexibilidad del software y las características del hardware reconfigurable. Los primeros resultados obtenidos han demostrado que la co-ejecución hardware/software de los algoritmos CNN sobre un HPRC alcanza una aceleración 10 veces superior a la obtenida por una solución software basada en PC convencional. Este resultado, aunque no ha sido todo lo satisfactorio que se esperaba, comparado con el obtenido por las implementaciones en VHDL a nivel RTL, sí pronostica un alto potencial de crecimiento que terminará por consolidar cuando las herramientas de síntesis maduren lo suficiente. Por ello, y dado que esta plataforma es especialmente útil para desarrollar algoritmos complejos sin exigir grandes conocimientos en electrónica, esta será considerada una de las principales líneas a desarrollar.

- Dentro de la línea anterior, la fusión de las redes CNN con otras técnicas *Soft Computing* como los algoritmos evolutivos pueden ser de interés para el estudio y el diseño de las plantillas utilizadas por las redes CNN. La combinación de dichas técnicas adquiere ahora una nueva dimensión gracias a la flexibilidad que proporciona el tándem formado por las platafomas HPRC y las herramientas de descripción a nivel ESL. La co-ejecución hardware/software implicaría que los algoritmos evolutivos, más complejos y con menor carga computacional, serían ejecutados en software, mientras que las funciones de evaluación a optimizar, constituidas por las redes CNN y más exigentes computacionalmente, serían ejecutadas en hardware.

Aportaciones

Las aportaciones de esta tesis doctoral se listan a continuación:

Publicaciones en revistas

J. Javier Martínez, F. Javier Toledo, E. Fernández, J. Manuel Ferrández "A retinomorphic architecture based on Discrete-Time Cellular Neural Networks using reconfigurable hardware", Neurocomputing, vol. 71, Elsevier, January 2008, pp. 766-775

J. Javier Martínez, F. Javier Toledo, E. Fernández, J. Manuel Ferrández, "Study of the contrast processing in the early visual system using a neuromorphic retinal architecture" Neurocomputing, vol. 72, Elsevier, January 2009, pp. 928-935

J. Javier Martínez, Javier Garrigós, F. Javier Toledo, E. Fernández, J. Manuel Ferrández "Implementation of a CNN-based retinomorphic model on a high performance reconfigurable computer", Neurocomputing, vol. 74, Elsevier, March 2011, pp. 1290-1297

Publicaciones en congresos internacionales

J. Javier Martinez, F. Javier Toledo, J. Manuel Vicente, "New emulated discrete model of CNN architecture for FPGA and DSP applications", Int. Work-Conference on Artificial and Natural Neural Networks (IWANN2003), June 2003, pp.33-40.

J. Javier Martínez, F. Javier Toledo, J. Manuel Ferrández, "Architecture implementation of a discrete cellular neuron model (DT-CNN) on FPGA", Microtechnologies for the New Millennium (SPIE 2005), May 2005, pp. 332-340.

F. Javier Toledo, J. Javier Martínez, F. Javier Garrigós, José Manuel Ferrández Vicente, "Image processing with CNN in a FPGA-based augmented reality system for visually impaired people". Int. Work-Conference on Artificial Neural Networks (IWANN 2005), June 2005, pp.906-912

J. Javier Martínez, F. Javier Toledo, F. Javier Garrigós, J. Manuel Ferrández,

"FPGA implementation of an area-time efficient FIR filter core using a self-clocked approach", Int. Conf. On Field Programmable Logic and Applications (FPL 2005), August 2005, pp. 547-550

J. Javier Martínez, F. Javier Toledo, J. Manuel Ferrández, "Discrete-time Cellular Neural network in FPGA", Field-Programmable Custom Computing Machines (FCCM 2007), April 2007, pp 293-294

J. Javier Martínez, F. Javier Garrigós, F. Javier Toledo, J. Manuel Ferrández "High performance implementation of an FPGA-based sequential DT-CNN", Int. Workconference on the Interplay between Natural and Artificial Computation (IWINAC 2007), LNCS 4528, June 2007, pp. 1-9

J. Javier Martínez, F. Javier Toledo, Javier Garrigós, E. Fernández, J. Manuel Ferrández "Model and hardware emulation of the first synapse of the retina using discrete-time cellular neural networks", IEEE International Conference on Image Processing (ICIP2009), November 2009, pp. 2705-2708

J. Javier Martínez, F. Javier Garrigós, F. Javier Toledo, J. Manuel Ferrández, "Using reconfigurable supercomputers and C-to-hardware synthesis for CNN emulation", Int. Work-Conference on the interplay between Natural and Artificial Computation (IWINAC 2009), June 2009, pp. 244-253.

J. Javier Martínez-Alvarez , Javier Toledo-Moreo, Javier Garrigós-Guerrero, J. Manuel Ferrandez-Vicente, "A Multi-FPGA Distributed embedded system for the emulación of multi-layer CNNs in real time video applications", Int. Workshop on Cellular Nanoscale Networks and Applications (CNNA 2010), February 2010.

J. Javier Martínez, Javier Garrigós , Isidro Villó, Javier Toledo, J. Manuel Ferrandez, "Hardware acceleration on HPRC of a CNN-based algorithm for astronomical images reduction", Int. Workshop on Cellular Nanoscale Networks and Applications (CNNA 2010), February 2010.

J. Javier Martínez-Alvarez , Javier Toledo-Moreo, Javier Garrigós-Guerrero, J. Manuel Ferrandez-Vicente, "An expandable hardware platform for implementation of CNN-based applications", Int. Work-Conference on the interplay between Natural and Artificial Computation (IWINAC 2011), May 2011, pp. 195-204

Bibliography

Almasi G. S. y Gottlieb A. (1989). *Highly parallel computing*. Benjamin-Cummings Publishing Co., Inc.

Almeida F., Giménez D., Mantas J. M. y Vidal A. M. (2008). *Introducción a la programación paralela*. Learning Paraninfo.

Anderson, M. G., D. (1992). *Artificial neural network techonology*. Data and analysis center for software.

Aomori H., kawakami K., otake T., takahashi N., yamauchi M. y tanaka M. (2005). Separable 2d lifting using discrete-time cellular neural networks for lossless image coding. En , *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, tomo 10, págs. 2607–2614.

Arena P., F. L.-M. G., Caponetto R. (1998). Cellular neural networks to explore complexity. En , *Soft computing, Springer Berlin*, tomo 1, págs. 120–136.

Avnet (2011). Line card. Http://www.avnet.com.

Balya D., Roska B., Roska T. y Werblin F. (2002). A cnn framework for modeling parallel processing in a mammalian retina. En , *International Journal on Circuit Theory and Applications*, tomo 30, págs. 363–393.

Bell C. G. y Newell A. (1971). *Computer structures: reading and Examples*. McGraw-Hill.

Boycott B., Hopkins J. y Sperling H. (1987). Cone connections of the horizontal cells of the rhesus monkey retin. En , *Proceedings of the Royal Society of London*, tomo 229, págs. 345–379.

Broomhead, L. D., D. (1988). Multivariable functional interpolation and adaptive networks. En , *Complex Systems*, tomo 2, págs. 321–355.

Brown R. y L., C. (1992). Chaos or turbulence. En , *International Journal of Bifurcation and Chaos*, págs. 1005–1009.

Buczynski R., Thienpont H., Jankowski S., Szoplik T. y Veretennicoff I. (1998). Programmable cnn based on optical thyristors for early image processing. En , *IEEE International workshop on Cellular Neural Networks*, págs. 259–264.

CANDY (2001). Candy, matcnn, aladdin. Http://cnn-technology.itk.ppke.hu/.

Carmona R., Jiménez G. F., Dominguez C. R., Espejo S., Roska T., Rekeczky C., Petras I. y Rodriguez V. A. (2003). A bio-inspired two-layer mixed-signal flexible programmable chip for early vision. En , *IEEE Transactions on Neural Networks*, tomo 14, págs. 1313–1336.

Carranza L., Jimenez F., Liñan G. y Roca E. (2005). Ace16k based stan-alone system for real-time pre-processing tasks. En , *SPIE the International Society for Optical Engineering*, págs. 872–879.

CELL (2001). Cell. Http://spctv1.eln.uniroma2.it/cell/.

Chua L., Roska T. y Venetianer P. L. (1993). The cnn is universal as the turing machine. En , *IEEE transactions on circuits and systems-part I*, tomo 40, págs. 289–291.

Chua L., R. T. (1993). The cnn paradigma. En , *IEEE transactions on circuits and systems-part I*, tomo 40, págs. 147–156.

Chua L., Y. L. (1988). Cellular neural network; part i: Theory; part ii: Applications. En , *IEEE Transaction on cicuits and systems*, tomo 35, págs. 1257–1290.

Chua L.O, R. T. (2002). *Cellular neural networks and visual computing foundations and applications*. Cambridge university press.

Cimagalli V., B. M. (1990). A neural network architecture for detecting moving objects. ii. En , *Cellular neural networks and their applications*, págs. 124–125.

Cimagalli V., B. M. (1993). Cellular neural networks: review. En , *Proceedings of sixth italian workshop on parallel architectures and neural networks*.

CNNsim (2001). Cnnsim. Http://www.isiweb.ee.ethz.ch/haenggi/CNNsim.html.

Collins, P. P., D.R. (1989). Considerations for neural network hardware implementations. En , *Circuits and Systems*, tomo 2, págs. 834–836.

Cray (2009). Web page. Http://www.cray.com/.

Dacey D., Packer O., Diller L., Brainard D., Peterson B. y Lee B. (2000). Center surround receptive field structure of cone bipolar cells in primate retina. En , *Vision Research*, tomo 40, págs. 1801–1811.

Dominguez C. R., Espejo S., Rodriguez V. A., Garcia V. I., Ramos J. y Carmona R. (1994). Sirena: a simulation environment for cnns. En , *Cellular Neural Networks and their Applications*, págs. 417–422.

Domínguez C. R., Espejo S., Rodríguez V. S., Carmona R., Foldesy P., Zarhdy A., Szolgay P., Szirinyi T. y Roska T. (1997). A 0.8 um cmos 2-d programmable mixed-signal focal-plane array-processor with on-chip binary imaging and instructions storage. En , *IEEE Journal of Solid State Circuits*, tomo 32, págs. 1013–1026.

Domínguez C. R., Espejo S., Rodríguez-Vazquez A. y Carmona R. (1994). A cnn universal chip in cmos technology. En , *International Workshop on Cellular Neural Networks and their Applications*, págs. 91–96.

Drasdo N., Millican C., Katholi C. y Curcio C. (2007). The length of henle fibers in the human retina and a model of ganglion receptive field density in the visual field. En , *Vision Research*, tomo 47, págs. 2901–2911.

Flynn M. J. (1966). Very high-speed computing systems. En , *Proc. of the IEEE*, págs. 1901–1909.

Furbe S. y Day P. (1992). Four phase micropipelined latch control circuits. En , *IEEE Transactions on Very Large Scale Integration (VLSI)*, tomo 4, nº 2, págs. 247–253.

George S. y Moschytz (1999). Analytic and vlsi specific design of robust cnn templates. En , *Journal VLSI Signal Processing Systems*, págs. 415–427.

Grossberg, S. (1976). Adaptive pattern classification and unversal recording; i. parallel development and coding of neural feature dectectors; ii. feedback expectation, olfaction, and illusions. En , *Biological Cybernetics*, tomo 23, págs. 23–121 187–202.

Hanggi M. y Moschytz G. (1997). Visualisation of cnn dynamics. En , *Electronics Letters*, tomo 20, págs. 1714–1716.

Harrer H. (1993). Multiple layer discrete-time cellular neural networks using time-variant templates. En , *IEEE Transactions on Circuits and Systems-part II*, tomo 40, págs. 191–199.

Harrer H., Nossek J., Roska T. y Chua L. (1993). A current-mode dtcnn universal chip. En , *IEEE International Symposium on Circuits and Systems*, tomo 4, págs. 135–138.

Harrer H., Nossek J. y Stelzl R. (1992). An analog implementation of discrete-time cellular neural networks. En , *IEEE Transactions on Neural Networks*, tomo 3, págs. 466–476.

Harrer H. y Nossek J. A. (1992). Discrete-time cellular neural networks. En , *International Journal of Circuit Theory and Applications*, tomo 20, págs. 453–467.

Herault J. (1996). A model of colour processing in the retina of vertebrates: From photoreceptors to colour opposition and colour constancy phenomena. En , *Neurocomputing*, tomo 12, págs. 113–129.

Hopfield, J. J. (1982). Neural networks and physcal systems with emergent collective computational abilites. En , *Natural academy of science*, tomo 79, págs. 554–558.

Hsin C., C. C.-T. L. C. C., Yung H. (2006). Image-processing algorithms realized by discrete-time cellular neural networks and their circuit implementations. En , *Chaos, Solitons and Fractals*, tomo 29, págs. 1100–1108.

Hwang K. y Briggs F. A. (1998). *Arquitectura de computadoras y procesamiento paralelo*. McGraw-Hill.

Intel (1991). *Intel 80170 NX electrically trainable analog neural network (ETANN)*. 80170 NX specification booklet, Intel Corp.

Jain, J. M. M.-K., A. K (1996). Artificial neural networks: a tutorial. En , *Computer*, tomo 29, págs. 31–44.

Johan A. H., suykens K. y joos V. (1996). Discrete time interconnected cellular nerual networks within nlq theory. En , *International Journal of Circuit Theory and Applications*, tomo 24, págs. 25–36.

Kenneth R. y Chua L. (1996). The cnn universal machine is as universal as a turing machine. En , *IEEE Transactions on Circuits and Systems-part I*, tomo 43, págs. 353–355.

Kohonen, T. (1972). Correlation matrix memories. En , *IEEE transaction on computers*, tomo 21, págs. 353–359.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. En , *Biological Cybernetics*, tomo 21, págs. 353–359.

Kolb H. (2003). How the retina works. En , *American Scientist*, tomo 91, págs. 28–35.

Kolb H., Fernandez E., Nelson R. y William B. (2011). Webvision. Http://webvision.med.utah.edu.

Kunng S. Y. (1988). *VLSI arrays processors*. Prentice Hall.

Liñán G. C. (2002). *Design of low-power mixed-signal programmable vision Chips*. Tesis Doctoral, Univ. of Seville.

Liñán G. C., Espejo S., Domínguez C. R. y Rodríguez V. A. (2002). Ace4k: An analog i/o 64x64 visual microprocessor chip with 7-bit analog accuracy: Research articles. En , *Int. J. Circuit Theory Appl.*, tomo 30, págs. 89–116.

Liñán G. C., Rodríguez V. A., Espejo Carranza L., Domínguez C. R. y Espejo M. S. (2003). A 1000 fps 128x128 vision processor with 8-bit digitized i/o. En , *Proc. 29th Eur. Solid-State Circuits Conf.*, págs. 61–64.

Manganaro, A. P. F. L., G. (1999). *Cellular neural networks, chaos, complexity and VLSI processing*. Springer.

Martínez J. J., Toledo J., Garrigos J. y Ferrández J. M. (2005). Filtro eficiente fir-mac para el diseño a alto nivel con xilinx system generator. En , *V Jornadas de Computación Reconfigurable y Aplicaciones, JCRA2005*, págs. 211–216.

Martínez J. J., Toledo J., Garrigos J. y Ferrández J. M. (2006). Convolución 3x3 de alta eficiencia para el diseño de aplicaciones de procesamiento de video a alto nivel. En , *VI Jornadas de Computación Reconfigurable y Aplicaciones, JCRA2006*, págs. 269–273.

Martínez J. J., Toledo J., Garrigos J. y Ferrández J. M. (2009). Using reconfigurable supercomputers and c-to-hardware synthesis for cnn emulation. En , *International Work-conference on the Interplay between Natural and Artificial Computation, IWINAC2009*, págs. 244–253.

Masland R. (2001). The fundamental plan of the retina. En , *Nature Neuroscience*, tomo 4, págs. 877–886.

Mika L., Poikonen J., Virta P. y Paasio A. (2008). A 64x64 cell mixed-mode array processor prototyping system. En , *Cellular neural networks and their applicaions*, págs. 1–1.

Minsky, P. S., M. L. (1969). *Perceptrons*. MIT Press.

Mirzai B., M. G. S. (1998). Influence of boundary conditions on the robustness of cellular neural networks. En , *IEEE transactions on circuits and systems-part I*, tomo 45, págs. 511–515.

Nagy Z., Voroshazi Z. y Szolgay P. (2005). An emulated digital retina model implementation on fpga. En , *International Workshop on Cellular Neural Networks and their Applications, CNNA*, págs. 278–281.

Nagy Z. (2007). *Implementation of emulated digital CNN-UM arquitectura on programmable logic devices and its applications*. Tesis Doctoral, University of Pannonia Veszprem, Hungary.

Nagy Z. y Szolgay P. (2003). Configurable multi-layer cnn-um emulator on fpga using distributed arithmetic. En , *Transactions on Circuits and Systems I: Fundamental Theory and Applications*, págs. 774–778.

Nagy Z. y Szolgay P. (2004). Emulated digital cnn-um implementation of a barotropic ocean model. En , *IEEE International Joint Conference on Neural Networks*, tomo 4, págs. :3137–3142.

Ortega J., Anguita M. y Prieto A. (2005). *Arquitectura de computadores*. Thomson.

Ortega S., Raygoza J. y Boemo E. (2003). Sincronización self-timed: protocolo de 4 fases. En , *Jornadas de Computación Reconfigurable y Aplicaciones, JCRA2003*, págs. 517–528.

P., T. (1993). Influence of boundary conditions on the behavior of cellular neural networks. En , *IEEE transactions on circuits and systems-part I*, tomo 40, págs. 207–212.

Paasio A., Dawidziuk A. y Porra V. (1997). Vlsi implementation of cellular neural network universal machine. En , *IEEE International Symposium on Circuits and Systems*, tomo 1, págs. 545–548.

Paasio A., Dawidziuk A. y Porra V. (1999). A qcif resolution binary i/o cnn-um chip. En , *J. VLSI Signal Process. Syst.*, tomo 23, págs. 281–290.

Paasio A., Kananen A., Halonen K. y Porra V. (1998). A 48 by 48 cnn chip operating with b/w images. En , *XXX*, tomo X, pág. XXX.

Paasio A., Kananen A., Laiho M. y Halonen K. (2002). An analog array processor hardware realization with multiple new features. En , *International Join Conference on Neural Networks, IJCNN*, págs. 1952–1955.

Packer O. y Dacey D. (2002). Receptive field structure of h1 horizontal cells in macaque monkey retina. En , *Journal of Vision*, tomo 2, págs. 272–292.

Pardo C. F. (2002). *Arquitecturas avanzadas*. Universidad de valencia.

Perko M., Fajfar I., Tuma T. y Puhan J. (1998). Fast fourier transform computation using a digital cnn simulator. En , *IEEE International Workshop on Cellular Neural Networks*, págs. 230–235.

Picocomputing (2011). Card products. Http://www.picocomputing.com.

Rodriguez-Vazquez, L.-C. G., A. Carranza L., Roca-Moreno E., Carmona-Galan R., Jimenez-Garrido F., Dominguez-Castro R. y Meana S. (2005). Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. En , *IEEE Transactions on Circuits and Systems I*, págs. 872–879.

Rodriguez-Vazquez A., D.-C. J. L. H. S.-S. E., Espejo E. (1993). Current-mode techniques for the implementation of continuosus and discrete time cellular neural networks. En , *IEEE transactions on circuits and systems-part II*, tomo 40, págs. 132–146.

Rosenblatt, R. (1962). *Artificial Intelligent*. Spartan Books.

Roska T., Bártfai G., Szolgay P., Szirányi A., Radványi A., Kozek T., Ugray Z. y Zarándy A. (1992). A digital multiprocessor hardware accelerator board for cellular neural networks cnn-hac. En , *Int. journal of circuit theory and applications*, tomo 20, págs. 589–599.

Roska T. y Chua L. O. (1993). The cnn universal machine: an analogic array computer. En , *IEEE transactions on circuits and systems-part II*, tomo 40, págs. 163–173.

Roska T., Kek L., Nemes A., Zarandy y Brendel M. (2000). *CNN Software Library (Templates and Algorithms), Version 1.1*. Analogic Computers Ltd.

Roska T. y Rodriguez V. A. (2000). Review of cmos implementations of the cnn universal machine-type visual microprocessors. En , *IEEE International Symposium on Circuits and Systems*, tomo 2, págs. 120–123.

Roska T., Szolgay P., Kozek T., Zarandy A., rekecky C., Ñemes L. y Kek L. (1997). *CADETWin: CNN application Develoment environment and toolkit under windows.*. MTA SZTAKI Budapes.

Roska T., C. L. O. (1992). Cellular neural networks with nonlinear and delay-type template elementes and non-uniform grids. En , *Int. j. circuit theory and applications*, tomo 20, págs. 469–482.

Roska T., T. P.-C. L., Boros T. (1990). Detecting simple motion using cellular neural networks. En , *Cellular neural networks and their applications*, págs. 127–138.

Rumelhart, M. J. L., D. E. (1986). *Parallel distributed processing: exploration in the microstructure of cognition*. MIT Press.

Sargeni F., Bonaiuto V. y Bonifazi M. (2006). Multiplexed circuit for star-cnn architecture. En , *IEEE International Workshop on Cellular Neural Networks and Their Applications*, págs. 1–5.

Schikuta E. y Weishäupl T. (2003). Cellular neural networks simulation on a top500 cluster. En , *Int.Conf. on Neural Networks and Applications*, págs. 763–769.

Silicon-Graphics (2009). Web page. Http://www.sgi.com.

Soos B., Rak A., Veres J. y Cserey G. (2008). Gpu powered cnn simulator (simcnn) with graphical flow based programmability. En , *Cellular Neural Networks and Their Applications*, págs. 163–168.

Sparso J. y Furber S. (2001). *Principles of asynchronous circuit design a systems perspective*. European low-power initiative for electronic system design, Dimes editor.

SRC-Computers (2009). Web page. Http://www.srccomp.com/.

Sriram S. y Bhattacharyya S. S. (2009). *Embedded multiprocessors: scheduling and synchronization*. CRC Press.

Sullivan O., Hegarty J., Kakizaki S., Kelly B. y McCabe E. (1996). A fully optically addressable connected component detector in cmos. En , *IEEE International workshop on Cellular Neural Networks*, págs. 439–443.

Szolgay P., Laszlo K., Kek L., Kozek T., Ñemes L., Petras I., Rekeczky C., Szatmari I., Z., Z. S. y Roska T. (1999). The cadetwin application software design system a tutorial. En , *Design automation day on cellular visual microporocessor*, págs. 58–68.

Tanenbaum A. S. (2000). *Organización de computadoras un enfoque estructurado*. Pearson educacion.

Tokes S., Orzó L., Ayoub A. y Roska T. (2003). Flexibly programmable opto-electronic analogic cnn computer (poac) implementation applying an efficient, unconventional optical correlator architecture. En , *Journal of Circuits, Systems and Computers*.

Tze-Yui H., Ping-Man L. y Chi-Sing L. (2008). Parallelization of cellular neural networks on gpu. En , *Pattern Recogn. Elsevier*, tomo 41, págs. 2684–2692.

Urtubia C. (2006). *Neurobiología de la visión*. Edicions UPC.

Vilariño D. L., Brea V. M., Cabello, D. y Pardo, J. M. (1998). Discrete-time cnn for image segmentation by active contours. En , *Pattern Recogn. Lett.*, tomo 19, págs. 721–734.

Vilariño, L. (2001). *Contornos activos a nivel de pixel: diseño e implementación sobre arquitecturas de redes no lineales celulares*. Tesis Doctoral, Dept. de electrónica y computación, universidad de santiago de compostela, españa.

Voroshazi Z., Kiss A., Nagy Z. y Szolgay P. (2008). Implementation of embedded emulated-digital cnn-um global analogic programming unit on fpga and its application. En , *International Journal of Circuit Theory and Applications*, págs. 589–603.

Weishupl T. y Schikuta E. (2004). How to parallelize cellular neural networks on cluster architectures. En , *Parallel Architectures, Algorithms, and Networks, International Symposium on*, tomo 0, pág. 439.

Wen C. Y. y Chung Y. W. (2000). The design of neuron-bipolar junction transistor (vbjt) cellularneural network (cnn) structure with multi-neighborhood-layer templates. En , *IEEE International Workshop on Cellular Neural Networks and Their Applications*, págs. 195–200.

Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behaviaral sciences*. Tesis Doctoral, Dept. of applied mathematics, Harvard University, Cambridge, Mass.

Willian M., Hart M. y Adler (1992). *Physiology of the Eye*. 9th ed. Mosby.

Xavier-de Souza S., Yalcin M. E., Suykens J. y Vandewalle J. (2004). Toward cnn chip-specific robustness. En , *IEEE Transactions on Circuits and Systems I*, tomo 51, págs. 892–902.

Xavier-de Souza S., Yalcin M. E., Vandewalle J., Johan A. K. y Joos V. (2003). Automatic chip-specific cnn template optimization using adaptive simulated annealing. En , *Proceedings of European Conference on Circuit Theory and Design*.

Xilinx (2004). Virtex-4 user guide, data sheet(ug070). Www.xilinx.com.

Xilinx (2008). Xtremedsp for virtex-4 (ug 073). Www.xilinx.com/support/.

Xilinx (2011). Fpga vs asic. Http://www.xilinx.com/company/gettingstarted/.

Yang T. (2002). *Handook of CNN image processing, all you nedd to kinow about celluar neural networks*. yang scientific press.

Yin C., Wan J., Lin H. y Chen W. (1999). The cloning template design of a cellular neural network. En , *Journal of The Franklin Institute, Elsevier*, tomo 336, págs. 903–909.

Zarandy A., Dominguez-Castro R. y Espejo S. (2002). Ultra-high frame rate focal plane image sensor and processor. En , *IEEE Sensors Journal*, tomo 2, págs. 559–565.

Zarandy A., Keresztes P., Roska T. y Szolgay P. (1998). An emulated digital architecture implementing the cnn universalmachine. En , *International Workshop on Cellular Neural Networks and Their Applications*, págs. 249–252.